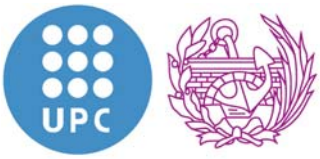


Escola Tècnica Superior d'Enginyers de Camins, Canals i Ports de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

PROJECTE O TESINA D'ESPECIALITAT

Títol

Análisis del problema del camino mínimo en redes urbanas

Autor/a

Borja Muñoz Echevarría

Tutor/a

M. Camino Balbuena Martínez

Departament

Matemàtica Aplicada

Intensificació

Transports

Data

Maig 2010

AGRADECIMIENTOS

Quisiera agradecer a mi tutora, Camino Balbuena, su gran ayuda durante el desarrollo de esta tesina. Gracias por su atención, correcciones y guía, sin los cuales no hubiera sido posible llevar a cabo este trabajo.

A mi familia, amigos y compañeros de trabajo por todo el apoyo y la ayuda que he recibido durante todo este tiempo.

RESUMEN

Título: *Análisis del problema del camino mínimo en redes urbanas*

Autor: Borja Muñoz Echevarría

Tutor: M. Camino Balbuena Martínez

El territorio moderno se ha de entender como un sistema de redes de transporte y comunicación. Los grafos representan una herramienta básica en el análisis de estos sistemas tan complejos, simplificando la realidad mediante nodos unidos por aristas.

Nuestro caso de estudio analiza el problema de búsqueda de caminos cortos. Estos problemas, conocidos como SPP (Shortest Path Problem) consisten en hallar el camino de mínimo coste desde un origen a un destino a través de una red conectada. El coste puede expresar o bien distancia, tiempo, o ambos a la vez.

Existe también el problema del camino más corto en un tiempo preciso (TCSPP), una generalización del SPP, donde se considera que un nodo de la red puede ser visitado sólo en un intervalo de tiempo determinado o ventana de tiempo. El objetivo de este tipo de problema es hallar el camino más corto que vaya del nodo origen al destino, ya sea visitando todos los nodos dentro de sus ventanas de tiempo respectivas o si no son visitados en el tiempo correspondiente, recibiendo una penalización finita, coste adicional o multa.

Otro problema clásico es el problema de la ruta de flujo máximo. En este caso, el problema consiste en encontrar el camino desde un nodo origen a un nodo destino que permita el paso de un flujo mayor. Para analizar este problema existe un famoso algoritmo que lo que hace precisamente es calcular todas las posibles rutas de penetración dentro de la red y da como solución el camino de flujo máximo del nodo origen al nodo destino. Este es el algoritmo de Ford y Fulkerson.

En esta tesina se quiere estudiar algunos algoritmos para resolver el problema clásico de encontrar la ruta entre un origen y un destino dados en una red de transporte, que permita hacer el recorrido en el menor tiempo posible.

El núcleo de la tesina es el algoritmo de Ford y Fulkerson, para encontrar la ruta entre un origen y un destino dados en una red de transporte, que permita hacer el recorrido transportando el mayor flujo posible.

Para ello, en primer lugar estudiaremos la teoría de grafos en general, y lo haremos mediante definiciones básicas de los elementos que la forman, definición de las tipologías de grafos existentes, y enunciado de distintos teoremas y proposiciones que nos serán útiles para alcanzar nuestro objetivo. El siguiente paso consistirá en analizar específicamente los elementos que forman los problemas de búsqueda de caminos, principalmente flujo y corte, y veremos tres algoritmos destacados para la resolución de este problema, el algoritmo de Dijkstra, el algoritmo de Floyd-Warshall, el algoritmo

de Bern y el algoritmo de Chen Yang. Por último, se procederá al análisis del algoritmo de Ford y Fulkerson. Definiremos el algoritmo y veremos un ejemplo detallado que nos sirva para entender su funcionamiento en profundidad.

Posteriormente y una vez entendido el algoritmo crearemos una aplicación con Visual Basic que sirva para desarrollar el algoritmo en cualquier red que definida previamente. Crearemos esta aplicación ejecutable puesto que no está a nuestro alcance ninguna aplicación en el mercado, al menos de forma gratuita, que ejecute este algoritmo para que pueda ser estudiado. Con la aplicación analizaremos algunos ejemplos para ver el funcionamiento del algoritmo en distintas redes.

ABSTRACT

Title: *Análisis del problema del camino mínimo en redes urbanas*

Author: Borja Muñoz Echevarría

Tutor: M. Camino Balbuena Martínez

The modern territory is to be understood as a system of transport and communication networks. The graphs represent a basic tool in analyzing such complex systems, simplifying reality through nodes connected by edges.

Our case of study analyzes the problem of finding short paths. These problems, known as SPP (Shortest Path Problem), consist of finding the least cost path from a source to a destination via a connected network. The cost can express either distance, time, or both.

It also analyzes the shortest path problem in a precise time (TCSPP), a generalization of the SPP, where a network node is considered to be visited only in a specified time or time window. The aim of this type of problem is to find the shortest path that goes from a source node to a destination node, either by visiting all nodes within their respective time windows, or, if they are not visited during the corresponding period, by receiving a finite penalty, additional cost or fine.

Another classical problem is the maximum flow path problem. In this case, the problem is to find a route from a source node to a destination node which allows the passage of a greater flow. To analyze this problem there is a famous algorithm whose task precisely to calculate all possible penetration routes into the network and gives as solution the maximum flow path from the source node to the destination node. This is the algorithm of Ford and Fulkerson.

The main objective of this thesis is to study some algorithms for solving the classical problem of finding the path between a given source and a given destination in a transport network which allows to make the journey in the shortest time possible.

Likewise, we will study the Ford and Fulkerson´ algorithm to find the route between a given source and a given destination in a transport network which allows to make the journey carrying the highest possible flow.

To do this, first we will study the theory of graphs in general, and we will do it through basic definitions of its elements, defining the types of existing graphs, and enunciating different sets of theorems and propositions that will be useful to achieve our goal.

The next step will be to analyze specifically the elements that make up the path-finding problems, mainly flow and cut, and we will look at three major algorithms for solving this problem, Dijkstra's algorithm, the Floyd-Warshall´ algorithm, the algorithm Bern and the algorithm of Chen Yang.

Finally, we will proceed to the analysis of Ford and Fulkerson´s algorithm. We will define the algorithm and look at a detailed example to help us to understand how it works in depth.

Subsequently, and having understood the algorithm, we will create a Visual Basic application that helps to develop the algorithm on any network which has been previously defined.

We will create this executable application because there aren’t any applications available on the market, at least for free, which runs this algorithm so that it can be studied.

With the application, we will look at some examples to see the performance of the algorithm in different networks.

ÍNDICE

1. Introducción y objetivos

El territorio moderno se ha de entender como un sistema de redes de transporte y comunicación. Los grafos representan una herramienta básica en el análisis de estos sistemas tan complejos, simplificando la realidad mediante nodos unidos por aristas.

Es por ello que queremos profundizar en el estudio de esta herramienta de la Ingeniería Civil, cuya aplicación es tan amplia que abarca cualquier tipo de coordinación de operaciones de transporte, abastecimiento de productos o planificación de procesos de todo tipo.

Nuestro caso de estudio analiza el problema de búsqueda de caminos cortos. Estos problemas, conocidos como SPP (Shortest Path Problem) consisten en hallar el camino de mínimo coste desde un origen a un destino a través de una red conectada. El coste puede expresar o bien distancia, tiempo, o ambos a la vez. Es un problema clásico y de gran importancia en la actualidad debido a sus múltiples aplicaciones y generalizaciones en redes de comunicaciones y redes de transporte.

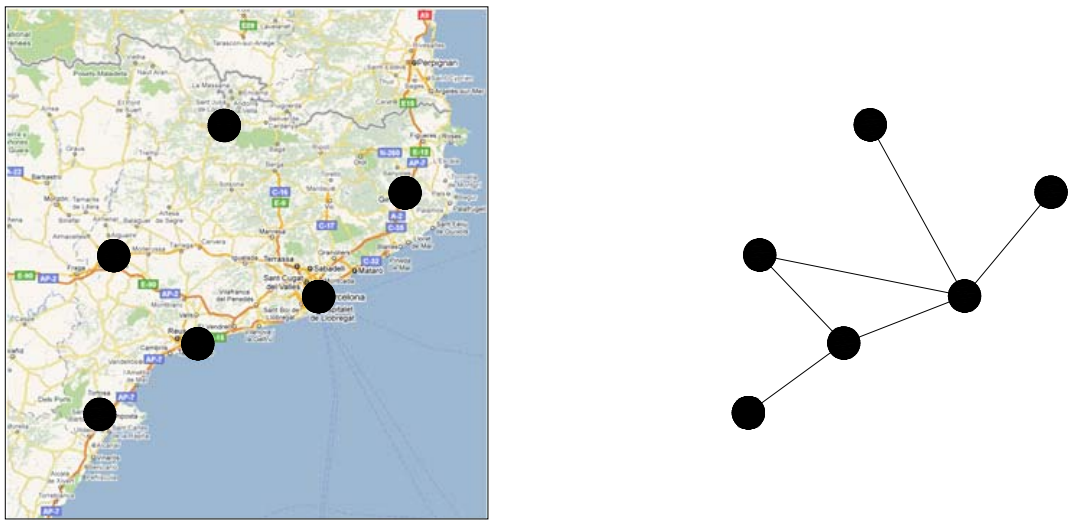


Figura 1.1. Simplificación de un territorio mediante nodos y el grafo resultante

También el problema del camino más corto en un tiempo preciso (TCSPP), una generalización del SPP, donde se considera que un nodo de la red puede ser visitado sólo en un intervalo de tiempo determinado o ventana de tiempo. El objetivo de este tipo de problema es encontrar el camino más corto que vaya del nodo origen al nodo destino, ya sea visitando todos los nodos dentro de sus ventanas de tiempo respectivas o si no son visitados en el tiempo correspondiente, recibiendo una penalización finita, coste adicional o multa

El problema del camino más corto en un tiempo preciso es perfecto para el análisis de redes urbanas, pues permite tener en cuenta las intersecciones reguladas por semáforos. Éstos tienen un ciclo rojo, verde y ámbar. Por tanto estos ciclos semafóricos hacen posible que en cada momento sólo una serie de rutas estén permitidas en la red de transporte.

Otro problema clásico es el problema de la ruta de flujo máximo. En este caso, el problema consiste en encontrar el camino desde un nodo origen a un nodo destino que permita el paso de un flujo mayor. Para analizar dicho problema existe un famoso algoritmo que consiste en calcular todas las posibles rutas de penetración dentro de la red y da como solución el camino de flujo máximo del nodo origen al nodo destino. Este es el algoritmo de Ford y Fulkerson.

El objetivo principal de esta tesina es estudiar algunos algoritmos para resolver el problema clásico de encontrar la ruta entre un origen y un destino dados en una red de transporte, que permita hacer el recorrido en el menor tiempo posible.

Del mismo modo, y como núcleo principal de la tesina, nos centraremos en el algoritmo de Ford y Fulkerson, para encontrar la ruta entre un origen y un destino dados en una red de transporte, que permita hacer el recorrido transportando el mayor flujo posible.

Para ello, en primer lugar estudiaremos la teoría de grafos en general, y lo haremos mediante definiciones básicas de los elementos que la forman, definición de las tipologías de grafos existentes, y enunciado de distintos teoremas y proposiciones que nos serán útiles para alcanzar nuestro objetivo.

El siguiente paso consistirá en analizar específicamente los elementos que forman los problemas de búsqueda de caminos, principalmente flujo y corte, y veremos tres algoritmos destacados para la resolución de este problema, el algoritmo de Dijkstra, el algoritmo de Floyd-Warshall, el algoritmo de Bern y el algoritmo de Chen Yang.

Por último, se procederá al análisis en profundidad del algoritmo de Ford y Fulkerson. Definiremos el algoritmo y veremos un ejemplo detallado que nos sirva para entender su funcionamiento exhaustivamente.

Posteriormente y una vez entendido el algoritmo crearemos una aplicación con Visual Basic que sirva para desarrollar el algoritmo en cualquier red que definida previamente.

Crearemos esta aplicación ejecutable puesto que no está a nuestro alcance ninguna aplicación en el mercado, al menos de forma gratuita, que ejecute este algoritmo para que pueda ser estudiado. Con la aplicación analizaremos algunos ejemplos para ver el funcionamiento del algoritmo en distintas redes.

2. Teoría de grafos

Como hemos comentado al inicio, para poder simplificar el complejo sistema de redes de transporte que nos rodea deberemos utilizar la teoría de grafos. Enmarcada dentro de la combinatoria, área de la matemática discreta, esta teoría permite modelar de forma simple cualquier sistema en el cual exista una relación binaria entre ciertos objetos; y es por eso que su aplicación tiene un carácter muy general.

El trabajo de Leonhard Euler, en 1736, sobre el problema de los puentes de Königsberg es considerado el primer resultado de la teoría de grafos. También se considera uno de los primeros resultados topológicos en geometría (no dependiente de ninguna medida). Este ejemplo ilustra la profunda relación entre la teoría de grafos y la topología. En el punto 2.1. de este apartado veremos este problema clásico al detalle.

En 1845 Gustav Kirchoff publicó sus leyes de los circuitos para calcular el voltaje y la corriente en los circuitos eléctricos.

En 1852 Francis Guthrie planteó el problema de los cuatro colores que plantea si es posible, utilizando solamente cuatro colores, colorear cualquier mapa de países de tal forma que dos países vecinos nunca tengan el mismo color. Este problema, que no fue resuelto hasta un siglo después por Kennet Appel y Wolfganag Haken, puede ser considerado como el nacimiento de la teoría de grafos. Al tratar de resolverlo, los matemáticos definieron términos y conceptos teóricos fundamentales de los grafos.

A continuación expondremos algunas definiciones básicas necesarias para poder profundizar en el tema.

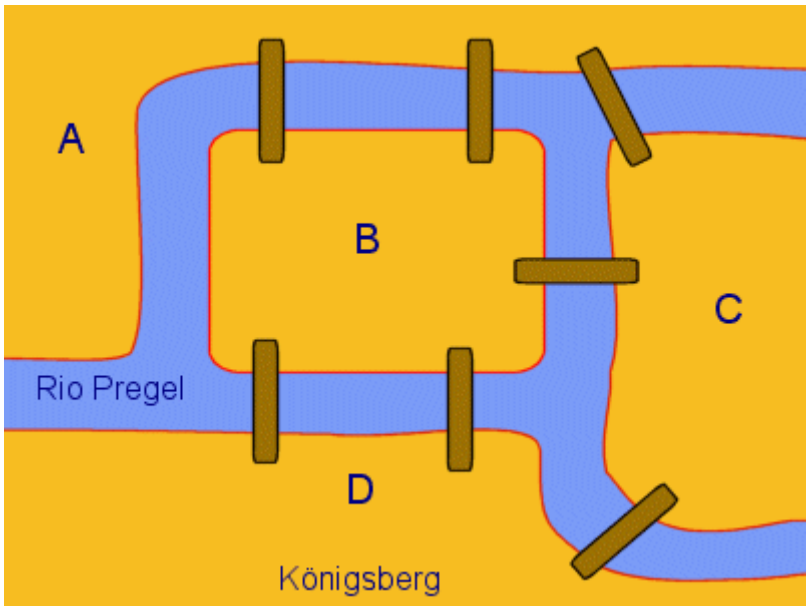
2.1. Problema de los Puentes de Königsberg

Como hemos comentado en la introducción, el origen de la teoría de grafos se asocia con la resolución que dio Leonard Euler (Basilea - Suiza, 1707-1783) del llamado problema de los puentes de Königsberg (1736). La resolución que dio Euler a este problema no solamente resolvía el problema, sino que introducía la noción de grafo y resolvía al mismo tiempo un problema de carácter más general.

Otro problema clásico es el de los puentes de Königsberg. Esta fue una populosa y rica ciudad de la Prusia Oriental. Hoy en día su nombre es Kaliningrado y pertenece a Rusia. Está situada en las orillas y en las islas del río Pregel, que en el siglo XVIII estaba atravesado por siete puentes. Es conocida por ser la cuna del filósofo I. Kant (1724-1804), pero en la historia de las Matemáticas es famosa por la disposición de sus puentes que dio lugar al problema que nos ocupa, precisamente en la época de Kant, que atrajo la atención de los más famosos matemáticos del momento.

El problema planteado consistía en planificar un paseo de manera que se cruzasen todos los puentes de la ciudad sin pasar por ninguno de ellos más de una vez. En la figura adjunta se aprecia la disposición de los 7 puentes.

Figura 2.9. Disposición de los 7 puentes a partir de un mapa de la época.



Figrua 2.10. Esquema simplificativo de la posición de los puentes.

La investigación que realizó Euler para resolver este problema fue presentada en 1736 en la Academia de Ciencias de San Pesterbusgo. La obra de Euler puede considerarse como el comienzo de la Teoría de Grafos, que forma parte de la Topología, rama de las Matemáticas que Leibniz llamó "geometría de la posición".
La idea de Euler fue representar la ciudad de Königsberg como un grafo en el que las cuatro regiones de tierra firme eran los vértices y los siete puentes eran las aristas.

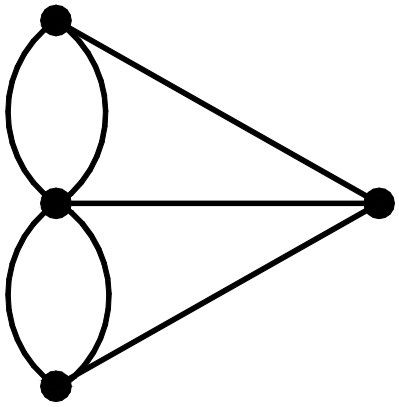


Figura 2.11. Grafo que representa los puentes de Königsberg

Así pues, el problema de los puentes de Königsberg pasa a ser un problema de teoría de grafos. La solución de dicho problema fue publicada por Euler en su artículo '*Solución de un problema relativo a la geometría de posición*'.

Una vez se dispone del grafo, debemos determinar si dicho grafo es euleriano, es decir, si contiene un circuito euleriano, que sería lo solución al problema.

Recordamos que, dado un grafo , un circuito es euleriano si recorre todo el grafo pasando sólo una vez por cada una de las aristas. Cuando el camino contiene a todas las aristas del grafo de manera que ninguna se repita y si comienza y termina en el mismo vértice.

Para determinar si el grafo es euleriano se requiere del siguiente teorema:

Teorema 2.4. Un grafo es *euleriano* y no tiene vértices aislados si, y sólo si ($\Leftrightarrow$), es conexo y el grado de todos sus vértices es par.

Demostración: Si es un grafo euleriano contiene un circuito euleriano y, al no tener vértices aislados, entonces cualquier par de vértices está conectado por la parte del circuito que va de uno a otro. Por tanto es conexo.
Por otra parte, como un circuito euleriano es un camino simple y cerrado que contiene a todas las aristas, por cada arista que llegue a un vértice debe haber otra que salga del mismo. Consecuentemente, el grado de cada vértice es un número par.

$\Leftarrow$) Partimos de que es conexo y todos sus vértices tienen grado par.

Si el número de aristas de es 1 ó 2 el resultado es inmediato. Procedemos por inducción: supongamos que tiene aristas y que el resultado es cierto para los grafos que cumplan las condiciones y tengan menos de aristas.

Usando que todo grafo en el que todos sus vértices tienen grado mayor o igual que dos posee un circuito tenemos que el nuestro contiene un circuito, digamos . Si contiene todas las aristas de , entonces es un circuito euleriano y hemos terminado. En caso contrario sea el grafo obtenido al suprimir de las aristas de y suprimir después los vértices que han quedado aislados. Puede que el grafo haya quedado dividido en subgrafos no conectados entre sí; cada uno de ellos es una componente conexa de .

Por haber eliminado las aristas de un circuito todos los vértices de tiene grado par. Por la hipótesis de inducción, cada componente conexa de contiene un circuito euleriano.

Como en cada componente conexa debe haber al menos un vértice de podemos obtener un circuito euleriano en (que es lo que queríamos conseguir) del siguiente modo:

Partimos de un vértice cualquiera de (que recordemos que no era un circuito euleriano). Recorremos hasta llegar a un vértice de una componente conexa $_i$ de Recorremos esta componente conexa a través del circuito euleriano que hemos visto que debe contener y continuamos recorriendo hasta que nos encontremos con un vértice de otra de las componentes conexas, realizando entonces la misma operación. Repetimos el procedimiento hasta llegar al vértice de partida, obteniendo así un circuito euleriano.

Observando ahora el grafo que habíamos obtenido de la ciudad de Königsberg y calculando el grado de todos sus vértices vemos que hay tres vértices con grado 3 y un vértice con grado 5. Es decir, en cada uno de los vértices del grafo correspondiente a los puentes de Könisberg concurre un número impar de aristas (3,5,3,3) con lo que no hay ningún vértice con grado par.

Así pues, según el teorema anterior, este grafo no contiene un circuito euleriano, llegando a la conclusión de que el problema planteado resulta imposible de resolver puesto no se puede empezar en un punto de la ciudad y recorrer cada uno de los puentes una sola vez y terminar en el punto de partida inicial.

2.2. Grafos y representación matricial

2.2.1. Definiciones básicas

Empezaremos estudiando los conceptos más básicos de la teoría de grafos.

Un *grafo simple* $G = (V, E)$ es un par formado por dos conjuntos, un conjunto finito no vacío cuyos elementos se llaman *nodos o vértices* $V = \{v_1, v_2, \dots, v_n\}$, y un conjunto de pares no ordenados de vértices distintos llamados *ramas o aristas* $E = \{a_1 = v_1v_2 = v_2v_1, a_2 = v_1v_3 = v_3v_1, \dots, a_m = v_{k-2}v_k = v_kv_{k-2}\}$.

Si la arista $a = \{u, v\} = uv$ relaciona los vértices u y v, se dice que estos son vértices *adyacentes* y también que el vértice u (o v) y la arista a son *incidentes*. De otro modo, los vértices se llaman *independientes*. Dos aristas son *independientes* si no tienen vértices en común. Análogamente, dos aristas diferentes son *adyacentes* si tienen un vértice en común.

Un *digrafo* o *grafo orientado* es un grafo direccionado, es decir, es un grafo en el que sus aristas tienen un sentido determinado y por ello se suelen denominar *aristas o arcos*. En este caso, si $a = (u, v) \in E$, decimos que el vértice u es *adyacente hacia* el vértice v y que v es *adyacente desde* u.

El *orden* de un grafo o digrafo es el número de nodos, $|V(G)| = n$, y el *tamaño* de un grafo o digrafo es el número de ramas o aristas respectivamente, $|E(G)| = m$. El tamaño del grafo cumplirá $0 < m < \frac{n}{2}$ puesto que el número máximo de aristas que puede tener un grafo se dará en el caso en el que todos los pares de nodos estén unidos por una rama.

El *grado de un nodo* en un grafo es el número de aristas que inciden en ese nodo. Dado $u \in V(G)$, $\Gamma(u)$ denota el conjunto de vértices adyacentes con u y su número, $d(u)$, es el grado del vértice u. El grado de un nodo en un digrafo puede ser de dos clases: el grado *de salida* o aristas que salen del nodo $d^+(u) = |\Gamma^+(\mathrm{u})|$, y el *grado de entrada* o aristas que llegan a ese nodo, $d^-(u) = |\Gamma^-(\mathrm{u})|$, siendo $\Gamma^+(\mathrm{u}) = \{\mathrm{w} \in$

$\mathrm{V}:(\mathrm{u},\mathrm{w}) \in \mathrm{E}\}$ y $\Gamma^{-}(\mathrm{u}) = \{\mathrm{w} \in \mathrm{V}:(\mathrm{w},\mathrm{u}) \in \mathrm{E}\}$ los conjuntos de vértices adyacentes desde y hacia u respectivamente. Se dice que el digrafo es regular si $\mathrm{d}^{+}(\mathrm{u}) = \mathrm{d}^{-}(\mathrm{u}) = \mathrm{d}$, para todo $u \in V$.

Teorema 2.1. Dado un grafo $G = (V, E)$, se cumple

$$\sum_{u \in V} d(u) = 2|E|$$

Este resultado se conoce como el lema del apretón de manos (handshacking lemma) porque se puede formular diciendo que en toda reunión de personas el número total de manos que se estrechan, cuando las personas se saludan entre ellas, es siempre par.

Corolario 2.1. En todo grafo *G* el número de vértices con grado impar es par.

Demostración. Separando en la suma $\sum_{u \in V} d(u)$ los términos correspondientes a vértices de grado par de aquellos correspondientes a vértices de grado impar se tiene:

$$2|E| = \sum_{d(u)\ par} d(u) + \sum_{d(u)\ impar} d(u)$$

Donde, siendo el primer sumando y la suma pares, el segundo sumando ha de ser también par.

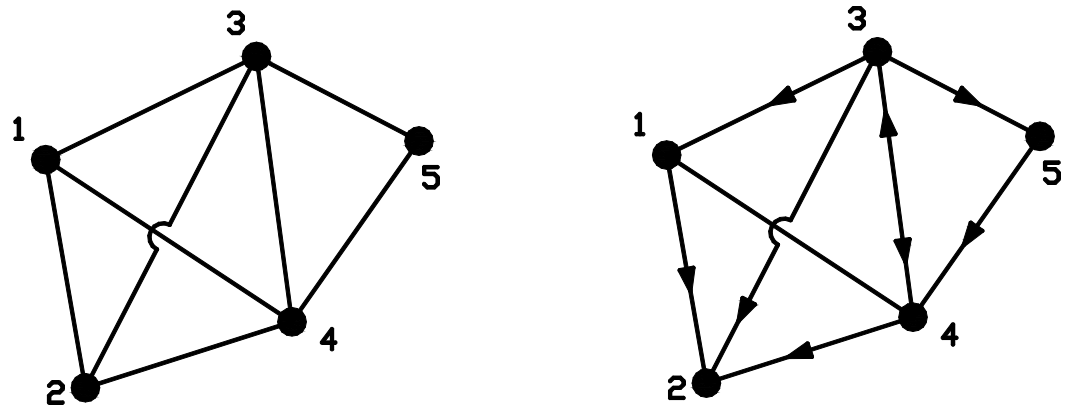


Figura 2.1. Vemos a la izquierda un grafo con 5 nodos y 8 aristas, y a la derecha el mismo grafo orientado, es decir, digrafo.

Uno de los grafos más interesantes por sus aplicaciones es el denominado *árbol*. Un grafo es un árbol cuando no contiene ciclos y el orden es igual al tamaño más 1. Cuando un digrafo es un árbol se denomina *árbol dirigido*. Un digrafo es un árbol dirigido cuando tiene un nodo raíz desde el cual hay un camino dirigido único hacia cada uno de los otros nodos.

Un *grafo o digrafo con peso* es un grafo dotado de una función de peso o coste que asigna un número real a cada una de las aristas. Obsérvese que todos los grafos y digrafos son grafos con peso. Basta asignar un 1 a las ramas o aristas en su caso.

Dado un grafo $G = (V, E)$, una secuencia de vértices $u_0, u_1, \ldots, u_l$ tales que $u_{i-1}u_i \in E$, $1 \leq i \leq l$, y $u_{i-1}u_i \neq u_{j-1}u_j$ si $i \neq j$ se denomina *recorrido* R *de longitud* l *entre* u_0 *y* u_l. Un *circuito* es un recorrido cerrado, es decir, un recorrido en el cual $u_0 = u_l$.

Un *camino* es un recorrido R en el cual todos los vértices son distintos. Un *ciclo* es un camino cerrado.

La *longitud* de un camino es el número de aristas. Se dice que un grafo o digrafo está *conectado* o que es *conexo* cuando dados dos vértices cualesquiera existe un camino que los une, y en este caso, la *distancia*, $d(u, v)$, entre dos vértices u y v es la longitud mínima de un camino entre estos vértices.

La *longitud* de un camino en un grafo con peso es la suma de los pesos de las aristas de ese camino. El camino mínimo entre dos nodos será el camino de peso mínimo entre dichos nodos, el cual representa la distancia entre esos nodos.

El *diámetro* del grafo o digrafo, $D = D(G)$ es la máxima de las distancias en él. La *excentricidad* de un vértice $u \in V(G)$ es la máxima de las distancias entre u y los otros vértices de G. Así, podemos definir el diámetro del grafo como la máxima de las excentricidades. Por otro lado, la mínima de las excentricidades se llama *radio*, $r(G)$.

2.2.2. Tipología de grafos

En primer lugar tenemos el *grafo simple*, definido anteriormente, donde no hay aristas orientadas, ni bucles, ni dos o más aristas entre un mismo par de vértices.

Además tenemos el *grafo dirigido o digrafo*, también definido anteriormente, donde hablaremos de inicio y final de una arista.

En el apartado anterior hemos visto dos ejemplos de grafo simple y digrafo.

Cuando en un grafo se permiten aristas múltiples, este se denomina *multigrafo*. Si además se permiten bucles, se denomina *pseudografo*. Ambos se pueden aplicar a digrafos.

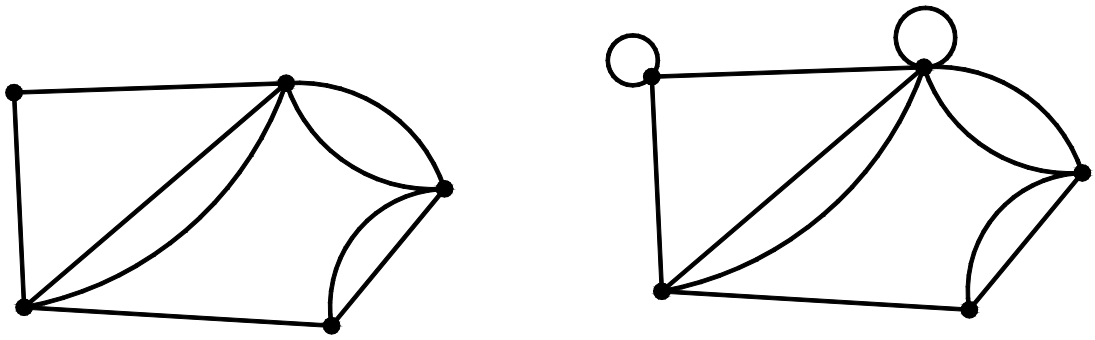


Figura 2.2. A la izquierda tenemos un multígrafo y a la derecha un pseudografo, con dos bucles.

El *grafo subyacente* de un grafo dirigido $G\,(V,E)$ es el grafo que se obtiene si se prescinde de la orientación de sus arcos.

Un *grafo conexo* es un grafo no dirigido de modo que, para cualquier par de vértices existe al menos un camino que los une.

Un *subgrafo* de un grafo $G = (V,E)$ es un grafo $G' = (V',E')$ cuyo conjunto de nodos y a aristas están incluidas en el conjunto de nodos y aristas de G. Así pues $V' \subseteq V \; y \; E' \subseteq E$. Cuando $V' = V$, el subgrafo G' se llama subgrafo generador de G.

Un *grafo nulo o vacío* no tiene ni vértices ni aristas.

Un *grafo trivial* es aquel que sólo tiene un vértice y no tiene ninguna arista.

Un *grafo completo* es aquel grafo en el que cada par de vértices están unidos por una arista, es decir, tiene todas las posibles aristas.

Un *grafo complementario* $G' = (V',E')$ es aquel grafo que contiene todas las aristas que le faltan al grafo $G = (V,E)$ para ser un grafo completo.

Ya hemos definido anteriormente que un *grafo acíclico*, es decir, que no contiene ciclos, se denomina *árbol*.

Un *grafo euleriano* es un grafo tal que existe un camino que contenga cada arco una sola vez. Para existir, el grado de cada uno de los nodos debe ser par y el grafo debe ser conexo.

Un *grafo hamiltoniano* es un grafo tal que existe un camino cerrado que pasa por todos los nodos una sola vez.

Un *grafo planar* es aquel que puede ser dibujado en el plano cartesiano sin cruce de aristas.

Un *grafo regular* es aquel con el mismo grado en todos los vértices. Si ese grado es k, lo llamaremos k-regular.

Un *grafo bipartito* es aquel grafo cuyo conjunto de vértices V se puede subdividir en dos subconjuntos V_1 y V_2 de manera que para cualquier par de vértices adyacentes (v_i, v_j) se verifica que $v_i \in V_1$ y $v_j \in V_2$. Es decir, si todas las aristas van de un subconjunto al otro.

Además, será un *grafo bipartito completo* si cada vértice de V_1 es adyacente sólo a cada vértice en V_2, y viceversa.

Un *digrafo es simétrico* si dados $u, v \in V(G)$ se tiene $(u, v) \in E(G)$ si y sólo si $(v, u) \in E(G)$. Cualquier resultado válido en grafos es automáticamente cierto en un digrafo simétrico.

Dos grafos simétricos G y H se llaman *grafos isomorfos* si existe una correspondencia biunívoca entre los vértices que preserva las adyacencias. Así pues, dos grafos isomorfos sólo se podrán diferenciar si sus vértices tienen distinta rotulación y, en general, por su representación gráfica. El orden, tamaño y el número de vértices con un grado determinado coincidirán en dos grafos isomorfos, son parámetros que denominamos invariantes por isomorfismo.

Por último, *los grafos platónicos* son los grafos formados por los vértices y aristas de los cinco sólidos regulares (sólidos platónicos), es decir, el tetraedro, el cubo, el octaedro, el dodecaedro y el icosaedro.

2.2.3. Representación matricial

Hemos dicho que un grafo o un digrafo puede visualizarse mediante un dibujo en que cada vértice se representa por un punto y cada arista o arco por una línea o línea dirigida respectivamente. No obstante, cuando se requiere el procesamiento por ordenador, resulta más conveniente disponer de representaciones matriciales del grafo o digrafo. Vemos a continuación las dos más importantes.

2.2.3.1.Matriz de adyacencia

Dado un grafo $G = (V, E)$ con n vértices $\{v_1, \ldots, v_n\}$ su matriz de adyacencia es la matriz de orden $n \times n$, $A(G) = (a_{ij})$ donde a_{ij} es el número de aristas que unen los vértices v_i y v_j. Se define como:

$$(A)_{ij} = \begin{cases} 1 & si \;\; v_i v_j \in E(G) \\ 0 & de\ otro\ modo \end{cases}$$

La matriz de adyacencia de un grafo es simétrica con elementos nulos en la diagonal. Si un vértice es aislado entonces la correspondiente fila (columna) está compuesta sólo por ceros. Si el grafo es simple (sin lazos ni aristas múltiples) entonces la matriz de adyacencia contiene solo ceros y unos (matriz binaria). Por otra parte, el número de elementos iguales a 1 en la fila (o columna) *i* de $A(G)$ es $d(v_i)$, el grado del vértice v_i, o lo que es equivalente, el número de caminos de longitud 1 que comienzan en el vértice v_i. De forma más general, las potencias de A dan información sobre los caminos en G, como veremos en el teorema siguiente.

Teorema 2.2. La entrada (i, j) de la potencia k de la matriz de adyacencia $(A^k)_{ij}$ es igual al número de recorridos (puede haber vértices y/o aristas repetidos) de longitud *k* entre v_i y v_j.

Demostración. Por inducción sobre *k*. La proposición se cumple si *k=1* por definición de A. Para $k > 1$:

$$(A^k)_{ij} = (A^{k-1}A)_{ij} = \sum_{l=1}^{n} (A^{k-1})_{il}(A)_{lj}$$

El término general de la suma anterior sólo es no nulo si $(A^{k-1})_{il} \geq 1 \;\; y \;\; (A)_{lj} = 1$. Pero en este caso, si $(A^{k-1})_{il} = m$, existen m recorridos de longitud $k-1$ entre $v_i \; y \; v_j$ que acaban con la arista $v_l v_j$. Como v_l es un vértice cualquiera, sumando sobre todo l se tiene el resultado.

Corolario 2.2. En un grafo conexo G, la distancia entre dos vértices $v_i \; y \; v_j$ es k si y sólo si k es el menor entero no negativo tal que $(A^k(G))_{ij} \neq 0$.

El método seguido para la construcción de una matriz de adyacencia a partir de un grafo es:

- Las columnas y filas de la matriz representan los nodos del grafo.
- Se llena la matriz de ceros (0).
- Por cada arista que une a dos nodos, se suma uno (1) al valor que hay actualmente en la ubicación correspondiente de la matriz.
- Si tal arista es un bucle y el grafo es no dirigido, entonces se suma dos (2) en vez de uno (1).
- Finalmente, se obtiene una matriz que representa el número de aristas (relaciones) entre cada par de nodos (elementos). Existe una matriz de

adyacencia única para cada grafo (sin considerar las permutaciones de filas o columnas), y viceversa.

A continuación se muestran dos ejemplos. En el primero vemos la matriz de adyacencia de un grafo y en el segundo vemos un ejemplo del Teorema 2.2.

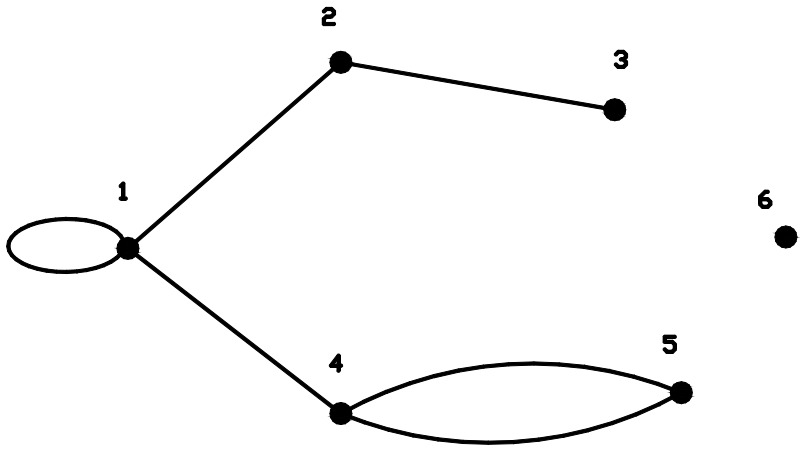
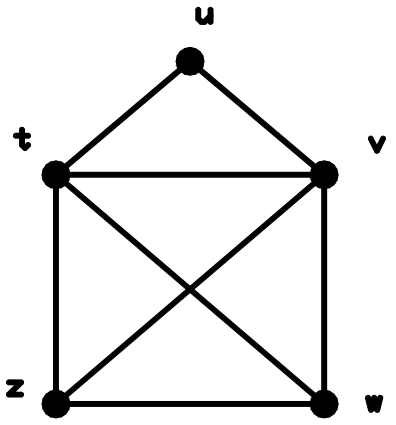


Figura 2.3. Grafo sobre el que construiremos la matriz de adyacencia

$$A(H) = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figura 2.4. Matriz de adyacencia del grafo anterior.



$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$$A^2 = \begin{pmatrix} 2 & 1 & 2 & 2 & 1 \\ 1 & 4 & 2 & 2 & 3 \\ 2 & 2 & 3 & 2 & 2 \\ 2 & 2 & 2 & 3 & 2 \\ 1 & 3 & 2 & 2 & 4 \end{pmatrix}$$

Figura 2.5. Matriz de adyacencia y potencia segunda de A del grafo

En el segundo ejemplo tenemos la matriz de adyacencia del grafo dibujado. Calculando A^2 vemos que, por ejemplo, el elemento $(A^2)_{25} = 3$ indica la existencia de

tres recorridos de longitud 2 entre los vértices $v_2 = v$ y $v_5 = t$: v,u,t; v,w,t y v,z,t. Por otra parte, la existencia de elementos nulos en la matriz A y el hecho que todos los elementos de A^2 sean diferentes de cero demuestra que el diámetro del grafo es 2.

2.2.3.2. Matriz de incidencia

Dado un grafo simple $G = (V,E)$ con $n = |V|$ vértices $\{v_1, \ldots, v_n\}$ y $m = |E|$ aristas $\{e_1,\ldots,e_m\}$, su matriz de incidencia es la matriz de orden nxm, $B(G) = (b_{ij})$, donde $b_{ij} = 1$ si v_i es incidente con e_j y $b_{ij} = 0$ en caso contrario.

Si G es un digrafo, $B(G)$ se define como

$$(B)_{ij} = \begin{Bmatrix} 1 & si \;\; v_i \; es\; incidentehacia\; el\; arco \;\; a_j \\ -1 & si \;\; v_i \; es\; incidente\; desde\; el\; arco \;\; a_j \\ 0 & de\; otro\; modo \end{Bmatrix}$$

La matriz de incidencia sólo contiene ceros y unos (matriz binaria). Como en el caso de la matriz de adyacencia, el número de unos en la fila i de B corresponde al grado del vértice v_i.

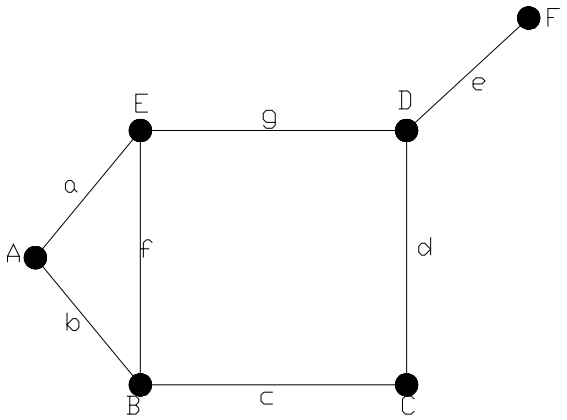
Una fila compuesta sólo por ceros corresponde a un vértice aislado.

A continuación veremos cómo construir una matriz de incidencia. Los pasos a seguir son los siguientes:

- Las columnas de la matriz representan las aristas del grafo.
- Las filas representan a los distintos nodos.
- Por cada nodo unido por una arista, ponemos un uno (1) en el lugar correspondiente, y llenamos el resto de las ubicaciones con ceros (0).

En el ejemplo de la figura, si sumamos las cantidades de 1′s que hay en cada columna, veremos que hay solo dos. Pero si sumamos las cantidades de unos 1′s que hay por cada fila, comprobaremos que los nodos B, D y E poseen un valor de 3. Ese valor indica la cantidad de aristas que inciden sobre el nodo.

A continuación vemos un ejemplo de matriz de incidencia con su grafo asociado.



$$B = \begin{pmatrix} a & b & c & d & e & f & g & \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & A \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & B \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & C \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & D \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & E \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & F \end{pmatrix}$$

Figura 2.6. Ejemplo de grafo con su matriz de incidencia

Las matrices de adyacencia y de incidencia de un grafo G representan su estructura y por ello han de estar relacionadas. Para expresar esta relación, sea D la matriz diagonal tal que $(D)_{ii}$ es el grado $d(v_i)$ del vértice i-ésimo de $V(G)$.

Teorema 2.3. Siendo B la matriz de incidencia, A la matriz de adyacencia y D la matriz diagonal,

$$BB^t = A + D$$

Demostración. Sea $V(G) = \{ v_1, v_2, \dots, v_n \}$ y $E(G) = \{ e_1, e_2, \dots, e_m \}$.

Si $i \neq j$,

$$(BB^t)_{ij} = \sum_{k=1}^{m} b_{ik} b_{jk} = a_{ij}$$

Ya que $b_{ik} b_{jk}$ sólo es diferente de 0 (y vale 1) si $e_k = v_i v_j$.

Si $i = j$,

$$(BB^t)_{ii} = \sum_{k=1}^{m} b_{ik} b_{jk} = d(v_i)$$

Ya que, ahora $\sum_{k=1}^{m} b_{ik} b_{jk}$ cuenta el número de aristas incidentes con v_i.

Las matrices $A(G)$ y $B(G)$ constituyen la base de las estructuras de datos más comúnmente utilizadas para representar un grafo o un digrafo en la memoria de un ordenador.

2.3. Circuitos y ciclos

Un *ciclo* es un camino finito en el que el nodo inicial corresponde con el final. En general se llama *circuito* a cualquier recorrido que comienza y termina en el mismo vértice.

2.3.1. Grafos eulerianos

Dado un grafo G, se dice que un circuito en G es euleriano si usa una única vez cada una de sus aristas. En el caso de que tal circuito exista, el grafo se denomina *grafo euleriano.* Del mismo modo, un recorrido que pasa una única vez por cada una de las aristas de G es un *recorrido euleriano*.

Para existir, el grado de cada uno de los nodos debe ser par y el grafo debe ser conexo, salvo vértices aislados. Estas dos condiciones son también suficientes para la existencia de circuitos eulerianos.

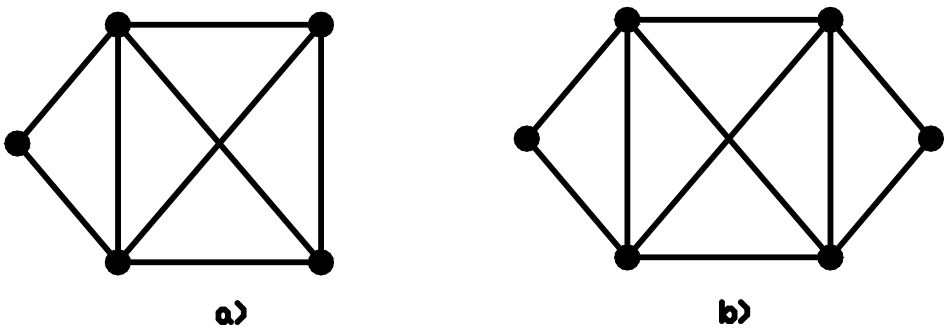


Figura 2.7. Grafo a) contiene recorridos pero no circuitos eulerianos, y el grafo b) contiene circuitos eulerianos.

2.3.1.1. Rompecabezas de Euler

El problema clásico con estos grafos se denomina Rompecabezas de Euler, también denominado recorrido o peregrinación del caballo de ajedrez. Euler resolvió este problema en 1759.

El problema consiste en pasar por todas las casillas del tablero usando el movimiento del caballo y sin pasar dos veces por la misma. Así pues, un caballo de ajedrez, que empieza sus movimientos desde la casilla número 1, puede pasar por las 64 casillas en orden numérico.

Una de las soluciones propuestas por Euler consiste en la secuencia de 64 jugadas con el caballo ejecutadas de forma que sólo pase una vez por cada casilla del tablero en la que además las filas y las columnas del recorrido suman 260 y, al detenerse a la mitad de cada una de las filas o columnas la suma resulta 130. Esta solución se conoce también como el cuadrado mágico.

8	1	48	31	50	33	16	63	18
7	30	51	46	3	62	19	14	35
6	47	2	49	32	15	34	17	64
5	52	29	4	45	20	61	36	13
4	5	44	25	56	9	40	21	60
3	28	53	8	41	24	57	12	37
2	43	6	55	26	39	10	59	22
1	54	27	42	7	58	23	38	11
	A	B	C	D	E	F	G	H

Figura 2.8. Cuadrado mágico. Solución propuesta por Euler.

En la Figura 2.8., el ciclo hamiltoniano se obtiene siguiendo la numeración de las casillas, de la número 1 a la 64.

Así pues, la solución del rompecabezas de Euler consiste en encontrar un ciclo Hamiltoniano en un grafo dónde las 64 casillas del tablero de ajedrez son los vértices del grafo y donde dos casillas son adyacentes si el caballo puede pasar de una a otra.

2.3.2. Ciclos hamiltonianos

Un camino hamiltoniano es un camino que visita todos los vértices exactamente una vez. Se denomina un ciclo hamiltoniano ó circuito hamiltoniano si es un ciclo que visita todos los vértices una sola vez y además el último vértice visitado es adyacente al primero (es decir, solo visita una sola vez cada vértice excepto el vértice del que parte y al cual llega).

Un grafo que contiene un ciclo hamiltoniano se denomina *grafo hamiltoniano***.**

Históricamente, el concepto hamiltoniano proviene de un juego inventado por el célebre matemático irlandés Sir William Rowan Hamilton, que propuso en 1886 el problema de encontrar un itinerario para recorrer veinte ciudades alrededor del mundo puestas en los vértices de un dodecaedro de manera que se pase una única vez por cada ciudad y se vuelva a la de salida. Esencialmente, se quiere encontrar un ciclo hamiltoniano del grafo formado por los vértices y las aristas de un dodecaedro.

Dicho grafo es, en efecto, hamiltoniano. Tal y como se muestra en la siguiente figura, los vértices están numerados de manera que, siguiéndolos en orden y uniendo el último con el primero, se obtiene un ciclo hamiltoniano.

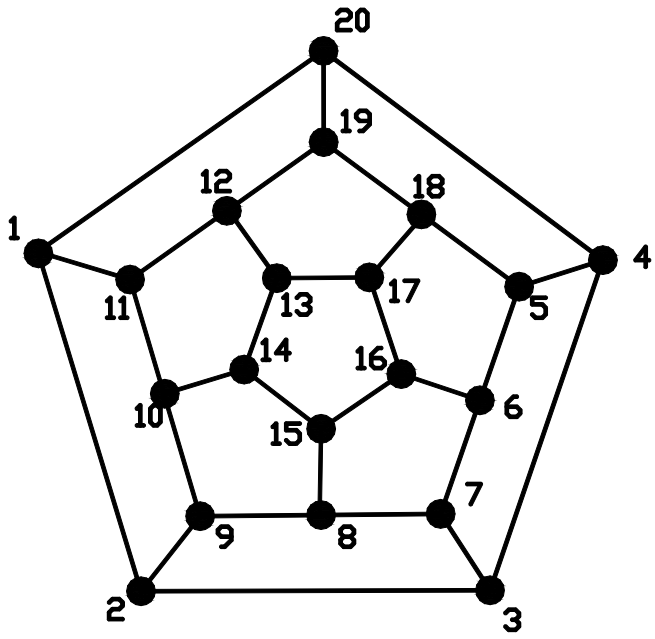


Figura 2.12. Ciclo en el grafo que representa la vuelta al mundo de Hamilton.

Las propiedades de ser hamiltoniano y de ser euleriano son próximas pero independientes. Una diferencia importante entre estas dos nociones es que, así como saber si un grafo es o no euleriano se responde con un teorema sencillo y definitivo en términos de la estructura del grafo, el problema de saber si es hamiltoniano resulta mucho más difícil; de hecho este es uno de los grandes problemas aún abiertos en la

teoría de grafos y hasta ahora no se conoce ningún resultado que dé condiciones necesarias y suficientes par a responder a esta cuestión, ya que esta cuestión responde como un problema NP-C (Non-Deterministic Polynomial-time-complete).

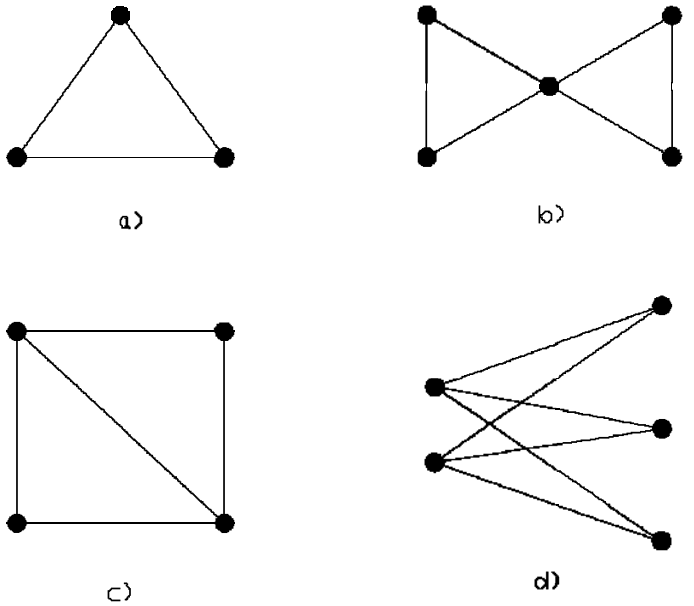


Figura 2.13. El grafo a) es euleriano y hamiltoniano; el grafo b) es euleriano y no hamiltoniano; el grafo c) es hamiltoniano y no euleriano; el grafo d) no es ni hamiltoniano ni euleriano.

Sin embargo existen algunos enunciados y teoremas que nos pueden dar condiciones suficientes o bien resultados específicos relativos a familias particulares de grafos.

Condiciones necesarias para que un grafo sea hamiltoniano:

1) ,
2) El grafo tiene que ser conexo
3) No debe existir ningún vértice de estrangulamiento, es decir, un vértice cuya supresión desconecta el grafo.

A partir de ahora supondremos que G(V,E) es un grafo que cumple todas las condiciones necesarias para ser hamiltoniano.

Un ejemplo de grafo hamiltoniano podría ser un grafo formado por los vértices y aristas de un cubo. Si es más grande o igual que 3 el grafo completo es hamiltoniano. De hecho, cualquier permutación de los vértices de da lugar a un ciclo hamiltoniando. El número de ciclos hamiltoniano en es entonces .

Obsérvese el siguiente lema, en el cual queda demostrada la idea de que si un grafo contiene un número suficiente de arista es más fácil poder recorrer un ciclo hamiltoniano.

Lema 2.1. Sea $G(V,E)$ un grafo de n vértices y sean v y u dos vértices no adyacentes tal que $\delta(u) + \delta(v) \geq n$. Entonces G es hamiltoniano si y sólo si $G + vu$ es hamiltoniano también.

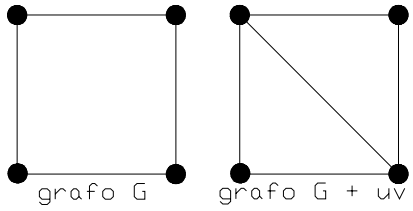


Figura 2.14. Representación gráfica de un grafo hamiltoniano, y del mismo ampliado uv.

Teorema 2.5.. Sea $G(V,E)$ un grafo de orden n con $n \geq 3$, si para cada par de vértices no adyacentes u, v de este grafo se satisface que $\delta(u) + \delta(v) \geq n$, entonces G es hamiltoniano.

Demostración. Teniendo en cuenta el lema anteriormente enunciado (si u y v son dos vértices no adyacentes, G es hamiltoniano si y sólo si $G + uv$ también lo es) y ampliándolo a todos los vértices no adyacentes, el teorema se puede reescribir de la siguiente manera: G es hamiltoniando si y sólo si el grafo completo k_n también lo es. Y esta claro que en un grafo completo k_n podemos encontrar un ciclo hamiltoniano.

Un teorema que en la práctica se utiliza para demostrar que un grafo plano no contiene ningún circuito hamiltoniano es el siguiente:

Una condición necesaria para la existencia de un circuito hamiltoniano viene dada por el siguiente teorema:

Teorema 2.7. Si el grafo $G(V,E)$ contiene un circuito hamiltoniano esto implica que para todo $V' \subset V$ $(V' \neq \emptyset)$ se verifica que el número de componentes del grafo $(G - V') \leq |V'|$.

Demostración. Si H es un circuito hamiltoniano de G tenemos que para todo V' el número de componentes del grafo $(H - V') \leq |V'|$. Pero además sabemos que $H - V'$ es un grafo parcial de $G - V'$ con lo que el número de componentes de $(G - V')$ es más pequeño o igual que el número de componentes de $(H - V')$.

Teorema 2.8. Todo grafo $G(V,E)$ completo tiene un ciclo hamiltoniano.

Demostración. Si el grafo es completo implica que cada par de nodos están conectados y al ser simétrico si $vu \in G$, también $uv \in G$; con lo que podríamos construir un ciclo $U = v_1, v_2, v_3, \dots, v_n$.

Teorema 2.9. Todos los grafos ciclos son hamiltonianos.

Demostración. Como ya hemos explicado anteriormente un grafo ciclo a aquel grafo que se asemeja a un polígono de n lados. Como el grado de cada vértice es 2 y el vértice v

tiene dos aristas incidentes $(v-1)v\ y\ (v+1)v$, podríamos crear un camino cerrado $U = v_1, v_2, v_3, \dots v_n \dots, v_1$ en el que no se repite ningún vértice a excepción del primero que aparece dos veces como principio y fin del camino.

Teorema 2.10. Todos los sólidos platónicos, considerados como grafos, son hamiltonianos. Como sólidos platónicos se entienden los poliedros regulares convexos.

Teorema 2.11. Si $G(V,E)$ es un grafo de orden n con $n > 2\ y\ \delta(v_i) \geq \frac{n}{2}$ para todo $v_i \in V$, entonces G es hamiltoniano.

Demostración. Suponemos que G cumple todas condiciones enunciadas y no contiene ningún circuito hamiltoniano. Sabemos que G no puede ser completo (ya que si lo fuese tendría un circuito hamiltoniano según el Teorema 2.8). Por lo tanto, podemos añadirle aristas hasta que podamos crear un circuito hamiltoniano $H = v_1, v_2, v_3, \dots . v_n$.

Definimos v_1 y v_n como vértices no adyacentes y los conjuntos de vértices $V' = \{v_i tal\ que\ (v_i, v_i + 1) \in E\}$ y $V'' = \{v_i\ tal\ que\ (v_i, v_n) \in E\}$.

Entonces $|V' \cap V''| = 0$, ya que si V' y V'' tuviesen algún vértice en común v_i , G tendría un circuito hamiltoniano $H' = (v_1, v_2, \dots v_i, v_n, v_{n-a}, \dots, v_{i+a}, v_1)$. Además $|V' \cup V''| < n$ ya que $v_n \notin V'$ ni de V". Si sumamos $\delta(v_1) + \delta(v_n) = |V'| + |V''| = |V' \cup V''| + |V' \cap V''| < n$ con lo que $\delta(v_1)$ o $\delta(v_n)$ es inferior a $n/2$ con lo queda contradicha la hipótesis del enunciado.

2.4. Flujos y conectividad

2.4.1. Redes de transporte

En ciertas aplicaciones interesa determinar el flujo máximo (de un fluido, datos, etc..) que fluye a través de una red (sistema de tuberías, sistema ferroviario, eslabones de comunicación, etc.). Mediante el modelo que proporciona la teoría de grafos podemos simular una estructura en la cual hay un cierto flujo que se transmite (petróleo, vagones, unidades de mensajes, etc.) a través de una red de enlaces, desde un punto de origen o fuente, a un punto final de salida o sumidero, pasando por diferentes nodos de recepción y de redistribución donde cada enlace que llega o sale de un nodo tienen una capacidad máxima de flujo.

Para ello definimos las redes de transporte. Se define *red de transporte* como $X = (G, s, t, c)$, donde $G = (V, E)$ es un digrafo o grafo orientado. Este tiene dos vértices distinguidos $s, t \in V$ (origen y destino). Además, *c* es una función de capacidad sobre E que asigna a cada arco $a = (u, v)$ un valor entero positivo.

Definimos s como el vértice de entrada a la red; t , como el vértice de salida de la red; y definimos $c(a)$ como la cantidad máxima de flujo que puede circular en un arco dirigido.

2.4.1.1. Problemas con redes

En los problemas con redes los esfuerzos se han centrado en dar algoritmos aproximados y métodos heurísticos, consistentes en la construcción de una solución y el posterior intento de mejora sistemática de esta solución.

Para ejemplificar lo que es un problema en una red de transporte a continuación describimos dos ejemplos clásicos en que se busca el camino de mínimo coste en redes de transporte. En el apartado 3 profundizaremos en el problema de búsqueda del camino mínimo.

2.4.1.1.1. Problema del viajante de comercio

Este problema (también conocido como TSP ('Travelling Salesman Problem') es una variante al problema de encontrar un ciclo hamiltoniano en un grafo, es decir, encontrar ciclos de mínimo coste que recorran todos los nodos de la red sin repetir ninguno.

El problema consiste en que un viajante tiene que visitar un conjunto de ciudades de forma que pase solamente una vez por cada una y que el trayecto total realizado se mínimo. El modelo que se utiliza para representarlo es un grafo completo dirigido con pesos en las aristas que representen las distancias (suponemos por ello que las aristas nunca toman valores negativos) y donde los nodos representan las ciudades.

Entonces se desea encontrar un ciclo hamiltoniano tal que la distancia sea mínima (o que la suma de los pesos de las aristas sea mínima). Este problema es un NP-completo, con lo que no existe un algoritmo eficiente (ni siquiera se sabe si existe) y solo algoritmos aproximados.

El problema simula cual debería ser el recorrido de un viajante de comercio que desea salir de una ciudad de origen 0 y visitarlas todas una vez, recorriendo una mínima distancia. Para encontrar la solución óptima se deberían examinar las $(n-1)!$ permutaciones de ciudades posibles (ya que hemos fijado una ciudad de origen).

Algoritmo del problema. Si el coste en un grafo completo de orden n satisface la desigualdad triangular $(c_{ij} \leq c_{ij} + c_{ik}\ , \forall v_i, v_j, v_k \in V$, donde c_{ij} son los costes en las aristas) debemos:

1- Seleccionar un vértice inicial ' v_1' y formar el ciclo C_1 (compuesto por el vértice origen)
2- Una vez obtenido C_1, buscamos el vértice v_i / $d(v_i, v_1)$ sea mínima y creamos el ciclo C_2 ($C_2 = v_1 v_i, v_1$)
3- Una vez obtenido el ciclo C_k, $1 \leq k \leq n-1$, calcular para cada $v \notin C_k$, $d(v, C_k) = \min\{d(v,u), u \in C_k\}$
4- Seleccionar un vértice $v_k \notin C_k$ tal que $d(v_k, C_k) = \min\{d(v, C_k), v \notin C_k\} = d(v_k, u_k)$
5- Formar el ciclo C_{k+1}insertando v_k inmediatamente antes de u_k en C_k.
6- Si $k+1 = n$, stop. Sino ir a 2.

Veamos un ejemplo de este problema:

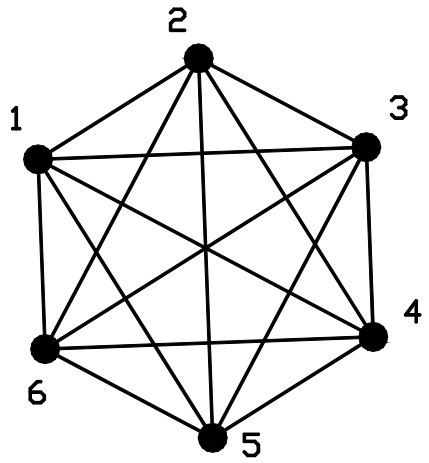


Figura 2.15. Grafo que representa el problema del viajante

vértices	1	2	3	4	5	6
1	0	3	3	2	7	3
2	3	0	3	4	5	5
3	3	3	0	1	4	4
4	2	4	1	0	5	5
5	7	5	4	5	0	4
6	3	5	4	5	4	0

Tabla 2.1. Tabla de coste del grafo para el algoritmo del TSP

1- Comenzaremos con el ciclo C_1 que está formado por el vértice de origen 1
2- El C_2 está formado por el vértice de C_1 y aquel vértice que esté a menor distancia (en este caso el vértice numero 4) C_2: 141.
3- Para formar el ciclo C_3 buscaremos aquellos vértices que estén a menor distancia de los que forman C_2 (en este caso el de menor distancia es 3) y lo colocaremos justo delante de aquel vértice al que la distancia es menor. C_3: 1341
4- Continuamos haciendo lo mismo, para hacer C_4 buscamos el vértice que este a menor distancia de alguno de los vértices que forman C_3, y que la distancia de ese vértice al resto sea la mínima posible, o sea que cogemos 2. C_4:12341
5- C_5:123461
6- C_6:1234561

El coste de este ciclo es $c(C_6)$= 3+3+1+5+4+3=19

El problema tiene considerables aplicaciones prácticas, aparte de las más evidentes en áreas de logística de transporte, que cualquier negocio de reparto, pequeño o grande, conoce. Por ejemplo, en robótica, permite resolver problemas de fabricación para minimizar el número de desplazamientos al realizar una serie de perforaciones en una plancha o en un circuito impreso. También puede ser utilizado en control y operativa optimizada de semáforos, etc.

2.4.1.1.2. Problema del cartero chino

Este problema (también conocido como CPP, 'Chinese Postman Problem'), fue planteado por primera vez por el matemático chino Kwan Mei-ko en 1962. El problema consiste en lo siguiente:

Un cartero tiene que recoger la correspondencia de la oficina de correos, entregarla en cada una de las calles de su zona y finalmente volver a la oficina. La pregunta que nos planteamos es qué ruta ha de seguir para poder recorrer todas las calles de forma que la distancia total recorrida sea mínima, y si es posible pasar sólo una vez por cada una de ellas.

Se puede clasificar este problema de tres modos distintos. El problema clásico del cartero suponía que el grafo era no orientado. Pero también podemos tener grafos con

todas o algunas de las aristas orientadas. Existen diversos algoritmos para resolver el problema.

2.4.1.1.3. Problema del cartero para grafos no orientados

Vamos a ver cómo se resuelve el problema en el caso en que tengamos un grafo no orientado $G(V,E)$. Obviamente, si G contiene un circuito euleriano, éste es el más corto ya que utiliza cada arco sólo una vez. Si G no es euleriano, entonces por lo menos una arista ha de ser recorrida más de una vez, reduciendo la productividad. Sea $a(i,j)$ la longitud de la arista (i,j) y $f(i,j)$ el número de veces que el cartero debe repetir la arista. Construimos un nuevo grafo $G*=(V,E*)$ que contenga $f(i,j)+1$ copias de cada arista (i,j). Se seleccionan los valores de $f(i,j)$ de manera que $G*$ tenga todos los vértices de grado par y que $\sum a(i,j)f(i,j)$ sea mínimo.

Si el vértice u es de grado impar en G, el número de aristas incidentes repetidas ha de ser impar. Si el grado es par, se ha de repetir un número par de aristas. Si empezamos el mayor camino posible a partir de un vértice de grado impar en G, acabaremos en un vértice de grado impar. En consecuencia, el problema puede transformarse en otro equivalente, que consiste en encontrar el conjunto de caminos entre parejas de nodos asimétricos (grado impar), cuya distancia total es menor. Para ello podemos utilizar por ejemplo el algoritmo de Floyd-Warshall, que se explicará más adelante.

2.4.1.2. Redes semafóricas

Parece intuitivo pensar que el mejor modelo para representar la circulación en una red urbana donde se desplazan vehículos es un digrafo, ya que las calles tienen un sentido determinado, que puede ser único o doble. Así que una red urbana es modelada por un digrafo, donde los nodos son las intersecciones de las calles y las aristas las propias calles. En este caso, los pesos de las aristas pueden corresponder al tiempo de viaje entre los nodos extremos de dicha arista, y en consecuencia, en este caso el coste es una asignación numérica positiva.

En cada uno de los accesos de las intersecciones se colocan semáforos, en cuya cabeza aparecen tres luces (roja, verde y ámbar) que se encienden sucesivamente. Las combinaciones de luces rojas y verdes definen todos los estados posibles en el conjunto de los semáforos de la intersección, que se denominan *fases*. Todas estas fases se van repitiendo sucesivamente y el tiempo necesario para que se pase por todas, junto con los períodos intermedios en que se encienden las luces ámbar, se denomina *ciclo del semáforo*. El *reparto* indica cómo se distribuyen en porcentaje los tiempos de rojo, verde y ámbar. El *tiempo de despeje* se produce cuando coexisten las luces rojas del semáforo en los accesos de la intersección. En teoría se trata de un tiempo de seguridad que permite pasar al último coche que ha pasado en verde.

Por su sencillez, los semáforos de tiempo fijo, es decir, con duración de ciclo y de fases fijo, es el más usado en la red urbana.

En la teoría de grafos, para resolver el problema de búsqueda de caminos de mínimo coste en un tiempo preciso (TCSPP), las ventanas de tiempo son una forma de representar los tiempos concretos, ya que indican cuál es el tiempo en que se puede visitar un nodo. Están definidas por un intervalo de tiempo, es decir, por el tiempo inicial y final en que dicho nodo está disponible.

Debido a las características de las redes semafóricas, las ventanas de tiempo que permiten modelizar una red regulada por semáforos son las de tipo 'hard', donde cada nodo debe ser visitado en su periodo de tiempo correspondiente. Si se visita el nodo fuera de su ventana de tiempo equivale a pasar un semáforo en rojo, por lo que la penalización es infinita.

Si en cambio el vehículo llega antes del inicio de su ventana de tiempo sufrirá un retraso o tiempo de espera antes de que pueda continuar su ruta.

Así, el uso de semáforos en las redes de transporte aumenta las restricciones de las mismas haciendo que se ajusten más a la realidad principalmente en el caso en que representamos redes urbanas.

2.4.2. Flujo y corte

Como vemos, el problema básico con redes consiste en determinar el valor del flujo máximo que se puede hacer llegar de s a t, a través de la red, así como la distribución de este por los diferentes enlaces.

Un flujo en R es una función f de valores reales definida en A que satisface la condición:

$$f^+(v) = f^-(v) \text{ para todo } v \in I$$

Donde $I = V\ \{s, t\}$, siendo s la fuente y t el destino de la red de transporte.

El valor $f(a)$ en un arco *a* es el flujo a lo largo de a debido al flujo f. La condición anterior requiere que, para cada vértice intermedio v, el flujo en la red en dirección a v es igual al flujo en la saliendo de v. Por esta razón, se denomina la *Condición de conservación.*

Un flujo f es *factible* si satisface, además, la *Restricción de capacidad:*

$$0 \leq f(a) \leq c(a) \text{ para todo } a \in E$$

La frontera superior en esta condición impone la restricción natural de que el ratio de flujo a lo largo de un arco no puede exceder la capacidad del arco. A lo largo de este capítulo, el término flujo siempre se refiere a uno que sea factible.

Toda red tiene al menos un flujo, porque la función f definida por $f(a) \coloneqq 0$, para todo $a \in E$, claramente satisface ambas condiciones.; se denomina la *función cero.*

Si X es un conjunto de vértices en una red R y f es un flujo en R, entonces $f^+(X) - f^-(X)$ se denomina el *flujo neto* de salida de X, y $f^-(X) - f^+(X)$ se denomina es *flujo neto* de entrada en X, relativo a f. La condición de conservación requiere que el flujo neto $f^+(v) - f^-(v)$ que sale de cualquier vértice intermedio sea cero. Así, es sencillo probar que, relativo a cualquier flujo f(x,y), el flujo neto $f^+(x) - f^-(x)$ de salida de x es igual al flujo neto $f^+(y) - f^-(y)$ de entrada a y. Esta cantidad se denomina valor de f. El valor de un flujo f debe ser expresado como el flujo neto de salida de cualquier subconjunto X de V tal que $x \in X$ y $y \in V \backslash X$, como veremos a continuación.

Proposición 2.1. Para cualquier flujo f en una red $R(x,y)$ y cualquier subconjunto X de *V* tal que $x \in X$ y $y \in V \backslash X$, el valor del flujo queda definido como

$$val(f) = f^+(X) - f^-(X)$$

Demostración. De la definición de flujo y su valor, tenemos

$$f^+(v) - f^-(v) = \begin{Bmatrix} val(f) & si & v = x \\ 0 & si & v \in X \ \backslash \{x\} \end{Bmatrix}$$

Sumando estas ecuaciones sobre X y simplificando, obtenemos:

$$val(f) = \sum_{v \in X} \left(f^+(v) - f^-(v) \right) = f^+(X) - f^-(X)$$

Un flujo en una red $R(x,y)$ es un *flujo máximo* si no hay ningún flujo en R de mayor valor. Los flujos máximos tienen mucha importancia en el contexto de las redes de transporte. Una red $R(x,y)$ que contiene un camino dirigido $(x \to y)$ con todos los arcos de capacidad infinita, evidentemente admite flujos de valor distintos. Sin embargo, estas redes no existen en la práctica, y consideramos todas las redes como redes de máximo flujo.

Un *corte* $K(s-t)$ en un digrafo $D(x,y)$ es un subconjunto de arcos tal que $(B, B^C) = \{(u,v);\ u \in B, v \in B^C\}$ donde $V = B \cup B^C$ tal que $s \in B\ y\ t \in B^C$. Igualmente se puede definir el corte inverso (B^C, B).

La capacidad $\boldsymbol{c((B, B^C))}$ de un $\boldsymbol{s-t}$ corte se define como la suma de las capacidades de los arcos que van de s a t y que no retroceden.

Veamos un ejemplo de corte.

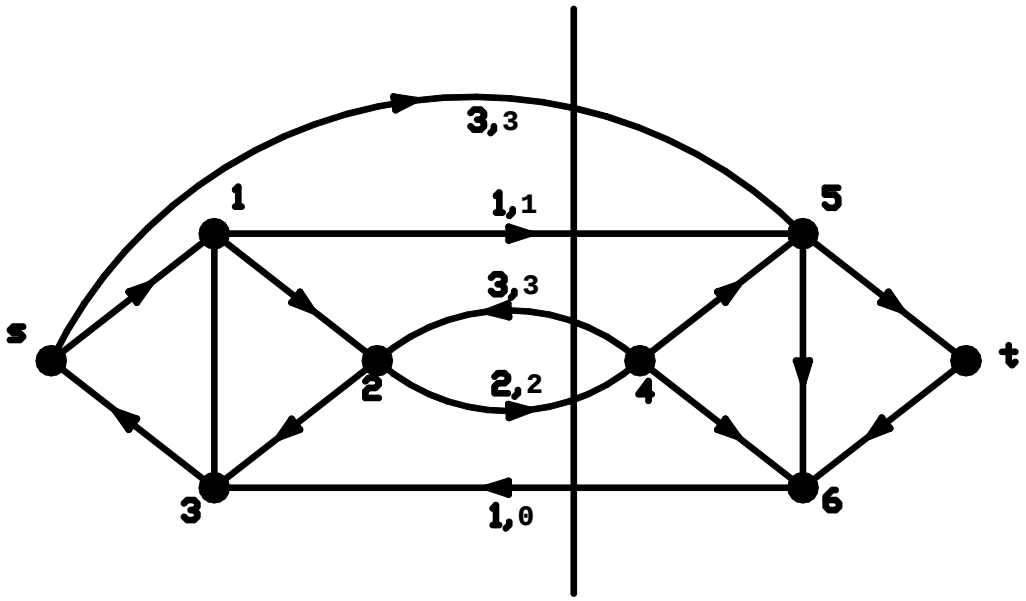


Figura 2.16. Grafo sobre el que realizamos el corte

$$B = \{s, 1,2,3\}\ Bc)\{t, 4,5,6\}$$

$$(B, B^C) = \{(s, 5), (1,5), (2,4)\}$$

$$c(B, B^C) = 3 + 1 + 2 = 6$$

$$\Phi(B, B^C) = 3 + 1 + 2 = 6$$

$$(B^C, B) = \{(4,2), (6,3)\}$$

$$c(B^C, B) = 3 + 1 = 4$$

$$\Phi(B^C, B) = 3 + 0 = 3$$

Flujos y cortes están relacionados de una manera simple: el valor de cualquier flujo (x, y) está limitado superiormente por la capacidad de cualquier corte que separe y de x. Para probar esta inecuación, conviene nombrar un arco *f-cero* si $f(a) = 0$, *f-positiva* si $f(a) > 0$, *f-no saturada* si $f(a) < c(a)$, *y f-saturada* si $f(a) = c(a)$.

Teorema 2.12. Para cualquier flujo f y cualquier corte K en una red R,

$$val(f) \leq cap(K)$$

Además, la igualdad se da sí y sólo si cada arco en $\delta^+(X)$ es f-saturado y cada arco en $\delta^-(X)$ es f-cero.

Demostración. Si $0 \leq f(a) \leq c(a)$,

$$(1) f^+(X) \leq c^+(X)\ and\ f^-(X) \geq 0$$

Así, aplicando la Proposición 2.1,

$$val(f) = f^+(X) - f^-(X) \leq c^+(X) = cap(K)$$

Tenemos que $val(f) = cap(K)$ sí y sólo si se da la igualdad en (1), es decir, sí y sólo si cada arco de $\delta^+(X)$ es f-saturado y cada arco de $\delta^-(X)$ es f-cero.

Un corte K en una red R es un *corte mínimo* si no existe ningún corte en R de menor capacidad.

Corolario 2.12. Sea f un flujo y K un corte. Si $val(f) = cap(K)$, entonces f es un flujo máximo y K es un corte mínimo.

Demostración. Sea f^* un flujo máximo y K^* un corte mínimo. Por el Teorema 2.5,

$$val(f) \leq val(f^*) \leq cap(K^*) \leq cap(K)$$

Pero, por hipótesis $val(f) = cap(K)$. Por consiguiente $val(f) = val(f^*)$ y $cap(K) = cap(K^*)$. Así f es un flujo máximo y K es un corte mínimo.

2.4.3. El teorema del flujo máximo – corte mínimo

Sea f un flujo en una red $R := R(x, y)$. A cada camino P en R (no tiene por qué estar dirigido), le asociamos un número entero no negativo $\epsilon(P)$ definido por:

$$\epsilon(P) := \min \{ \epsilon(a) : a \in A(P) \}$$

Donde

$$\epsilon(a) := \begin{cases} c(a) - f(a) & si \quad a\ es\ un\ arco\ de\ P\ hacia\ delante \\ f(a) & si \quad a\ es\ un\ arco\ invertido\ de\ P \end{cases}$$

$\epsilon(P)$ es la cantidad máxima en que podemos incrementar el flujo f a lo largo de P sin violar las restricciones. El camino P se denomina *f-saturado* si $\epsilon(P) = 0$ *y f-no saturado* si $\epsilon(P) > 0$, es decir, un camino *f-no saturado* es aquel que no está siendo usado a su máxima capacidad.

Un camino *f-incremental* es un camino *f-no saturado*. La existencia del mismo es importante pues implica que f no es un flujo máximo. Añadiendo un flujo adicional de $\epsilon(P)$a lo largo de P, obtenemos un nuevo flujo f' de valor mayor. Definimos $f' : A \rightarrow \mathbb{R}$ como:

$$f'(a) := \begin{cases} f(a) + \epsilon(P) & si\ a\ es\ un\ arco\ de\ P\ hacia\ delante \\ f(a) - \epsilon(P) & si\ a\ es\ un\ arco\ invertido\ de\ P \\ f(a) & en\ cualquier\ otro\ caso \end{cases}$$

Proposición 2.2. Sea f un flujo en una red R. Si existe un camino f-incremental, entonces f no es un flujo máximo. Concretamente, la función f' definida anteriormente es un flujo en R de valor $val(f') = val(f) + \epsilon(P)$.

Proposición 2.3. Sea f un flujo en una red $R \coloneqq R(x, y)$. Supongamos que no existe un camino f-incremental en R. Sea X un conjunto de todos los vértices alcanzables desde x por caminos f-no saturados, y sea $K \coloneqq \partial^+(X)$. Entonces f es un flujo máximo en R y K es un corte mínimo.

Demostración. Tenemos que $x \in X$. Además, $y \in V \setminus X$ porque no existe un camino f-incremental. Por tanto K es un corte en R.

Considerando un arco $a \in \delta^+(X)$, con cola u y cabeza v. Como $u \in X$, existe un camino Q f-no saturado. Si el arco a fuera f-no saturado, Q se podría extender con el arco a para producir un camino f-no saturado. Pero $v \in V \setminus X$, así que no existe tal camino. Por tanto, el arco a debe ser f-no saturado.

El mismo razonamiento se produce para $a \in \delta^-(X)$. En este caso se comprueba que el arco a debe ser f-cero. Y de nuevo por el Teorema 2.5, tenemos que $val(f) = cap(K)$. Su corolario implica que f es un flujo máximo en R y que K es un corte mínimo..

Como consecuencia de las dos proposiciones anteriores establecemos el siguiente teorema.

Teorema 2.13.. *El teorema de máximo flujo y mínimo corte*. En cualquier red, el valor de un flujo máximo es igual a la capacidad de un corte mínimo.

Demostración. Sea f un flujo máximo. Por la Proposición 2.2, no puede haber caminos f-incrementales. El teorema deriva directamente de la Proposición 2.3.

El teorema del flujo máximo y mínimo corte muestra que siempre podemos demostrar que un flujo es máximo simplemente encontrando un corte cuya capacidad sea igual al valor del flujo. Muchos resultados de la teoría de grafos son resultado directo de este teorema. El algoritmo de Ford-Fulkerson, que queremos estudiar en esta tesina, es uno de ellos.

3. Problema del camino mínimo

3.1. Introducción al problema

También conocido como SPP (Shortest Path Problem), es el problema principal en redes de transporte, donde simulamos una estructura en la cual hay un cierto flujo que se transmite a través de la red desde un centro de producción s hasta un centro destinatario t pasando por distintos nudos de recepción y de redistribución, donde cada enlace tiene una capacidad máxima de flujo.

El problema consiste en determinar el camino mínimo que hace llegar un flujo desde s hasta t. El problema del camino mínimo constituye uno de los problemas de optimización combinatoria más estudiados, debido a sus numerosas aplicaciones en diferentes campos.

Se han propuesto numerosos algoritmos para resolver este problema, en este apartado veremos algunos de ellos.

3.2. Árboles generadores

Un *árbol generador* de un grafo G es un subgrafo generador de G que es árbol. Notemos que, si T es un árbol generador de G, entonces T es un árbol maximal contenido en G en el sentido siguiente: si $e \in E(G)$ es una arista que no pertenece a $E(G)$, entonces $T + e$ ya es un subgrafo de G que contiene un ciclo. Veamos un ejemplo.

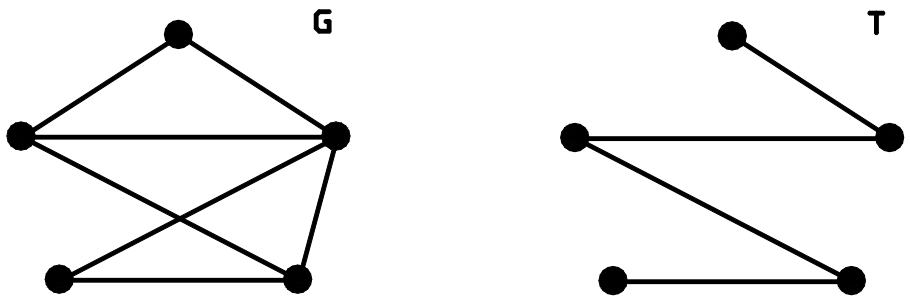


Figura 3.1. T es un árbol generador de G

Teorema 3.1. Todo grafo conexo tiene un árbol generador.

Demostración. Dado $G(V,E)$ conexo, sea $u \in V$ un vértice cualquiera. Para cada $v \in V$, $v \neq u$, escogemos un vértice w_v adyacente con v y tal que $d(u,w_v) = d(u,v) - 1$ (w_v puede coincidir con u). Sea $E' \subset E$ el conjunto de aristas de la forma $w_v v$ obtenido de esta manera. Entonces, el subgrafo $T = (V,E')$ es árbol generador de G. En efecto, por construcción existe en T un camino entre u y cualquier otro vértice v. Así, T es un subgrafo conexo. Por otra parte, si T tuviese un ciclo Γ y w fuese un vértice de Γ tal que la distancia $d(u,w)$ fuese máxima, entonces el vértice w sólo podría ser adyacente hacia otro vértice del ciclo, cosa que contradice el hecho de que en un ciclo cada vértice es adyacente hacia otros dos.

Con esta demostración, es sencillo formular un algoritmo para obtener un árbol generador de un grafo conexo. Además, el árbol generador T obtenido tiene la propiedad de preservar las distancias desde el vértice u. Es decir, $d_G(u,v) = d_T(u,v)$, para $v \in V(G)$.

3.3. Algoritmos básicos

La mayoría de los algoritmos propuestos en la literatura para resolver estos árboles mantienen, en cada paso, un árbol generador T de raíz r, y comprueban si las etiquetas de coste C que representan, o los costes de los caminos salientes de r a través del árbol o una cota superior de estos costes, satisfacen las condiciones de mínima distancia que se desean alcanzar.

En todos ellos se ha de determinar qué operación fundamental se realiza con el fin de poder evaluar la complejidad expresada como el número total de estas operaciones en relación con el volumen de los datos de entrada.

3.3.1. Algoritmo de Dijkstra

El *algoritmo de Dijkstra*, también conocido como el de *caminos mínimos* puesto que fue el primero de este tipo, determina el camino más corto dado un vértice origen al resto de vértices en un grafo con peso. Una vez encontrados todos los caminos mínimos el algoritmo se detiene. El algoritmo debe su nombre a Edsger Dijkstra, científico de la computación de origen holandés, quien lo describió por vez primera el año 1959.

El algoritmo determina en primera iteración el nodo más cercano v_k (en distancia, es decir, en peso) al origen v_0. A ese nodo lo etiqueta de forma permanente como v_k (distancia a origen, u_0). A los demás nodos se les asigna una etiqueta temporal. En definitiva, el algoritmo crea dos conjuntos P_j y T_j de nodos con etiqueta permanente y temporal respectivamente. La idea es que en cada iteración, partiendo del origen y vía los nodos $v_i \in P_j$ se determine el siguiente nodo más cercano al origen para pasarlo

del conjunto T_j al P_j. Así hasta que se etiqueta el último nodo de forma permanente y el algoritmo se detiene.

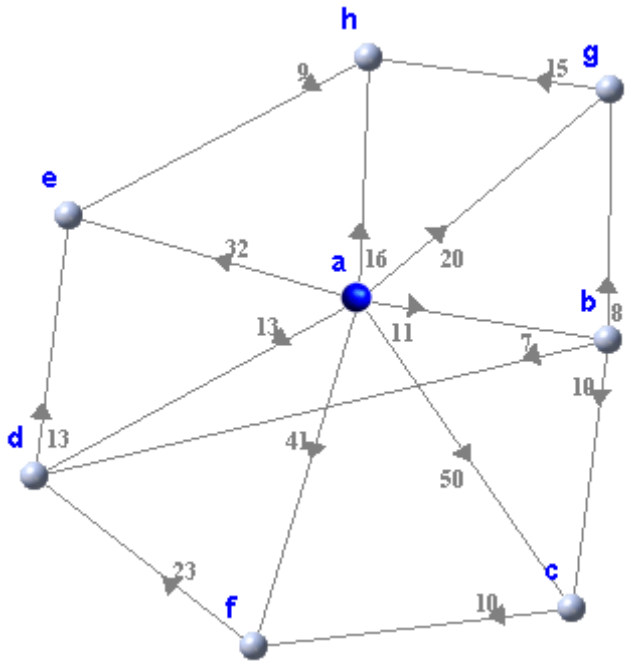


Figura 3.2. Situación inicial de la red

Veamos un *ejemplo* de aplicación de este algoritmo:

Una central de barrio u_0 de una empresa de alarmas, da servicio a siete domicilios de ese barrio. Se dispone de información sobre la red (arcos que conectan la central con los domicilios entre ellos). Lo que le interesa a la empresa es averiguar cuál es el camino más rápido desde la central a cualquiera de los siete domicilios a los que sirve.

1ª iteración: buscamos el camino de coste mínimo desde el nodo origen a. Este camino es el que va de a a b, por lo que b queda etiquetado de forma permanente como $b \rightarrow (11, a)$. Los otros nodos quedan etiquetados de forma temporal como $v_i \rightarrow (d_{va}, a)$. Así, los conjuntos de nodos con etiqueta permanente y temporal quedan de la siguiente forma:

$$P_1 = \{a, b\}\ T_1 = \{c, d, e, f, g, h\}$$

2ª iteración: buscamos el nodo más cercano al origen después de b teniendo en cuenta caminos directos pero también caminos pasando por b. Este nodo es el d que queda etiquetado permanentemente como $d \rightarrow (13, a)$. Los conjuntos de nodos con etiqueta permanente y temporal quedan de la siguiente forma:

$$P_2 = \{a, b, d\}\ T_2 = \{c, e, f, g, h\}$$

3ª iteración: buscamos el nodo más cercano al origen después de b y d teniendo en cuenta caminos directos y también caminos pasando por b y d. Este nodo será el h que queda etiquetado permanentemente como $h \rightarrow (16, a)$. Los conjuntos de nodos con etiqueta permanente y temporal quedan de la sigueitne forma:

$$P_3 = \{a, b, d, h\}\ T_3 = \{c, e, f, g\}$$

4ª iteración: buscamos de nuevo el nodo más cercano al origen después de b, d y h teniendo en cuenta también caminos que pasen por b, d y h. Este nodo es el nodo g que queda etiquetado permanentemente como $g \rightarrow (19, b)$. Los conjuntos de nodos con etiqueta permanente y temporal quedan de la siguiente forma:

$$P_4 = \{a, b, d, g, h\}\ T_4 = \{c, e, f\}$$

5ª iteración: buscamos el nodo más cercano al origen después de b, d, g y h teniendo en cuenta los caminos directos y también aquellos que pasen por b, d, g y h. Este nodo será el c que queda etiquetado permanentemente como $g \rightarrow (21, b)$. Los conjuntos de nodos con etiqueta permanente y temporal quedan de la siguiente forma:

$$P_5 = \{a, b, c, d, g, h\}\ T_5 = \{e, f\}$$

6ª iteración: buscamos el nodo más cercano al origen después de b, c, d, g y h teniendo en cuenta caminos directos pero también caminos que pasen por b, c, d, h y g. Este nodo es el nodo e que queda etiquetado permanentemente como $e \rightarrow (27, h)$. Los conjuntos de nodos con etiqueta permanente y temporal quedan de la siguiente forma:

$$P_6 = \{a, b, c, d, e, g, h\}\ T_6 = \{f\}$$

7ª iteración: buscamos el camino mínimo que nos lleve a f desde el origen teniendo en cuenta los caminos directos pero también caminos pasando por b, c, d, e, h y g. El camino mínimo es el que pasa por b y c así que f queda etiquetado permanentemente como $f \rightarrow (31, c)$. Los conjuntos de nodos con etiqueta permanente y temporal quedan de la siguiente forma:

$$P_7 = \{a, b, c, d, e, f, g, h\}\ T_7 = \{-\}$$

A continuación se muestra el proceso iterativo de forma gráfica. En las figuras, los arcos de color rojo son caminos candidatos a ser el mínimo mientras que los de color amarillo son el camino finalmente seleccionado. Al final, se obtiene una figura con todos los arcos de color amarillo que muestra el camino de mínimo coste para ir desde el origen (la central de alarmas) a cualquiera de los otros siete nodos (los domicilios a los que sirve).

A continuación vemos gráficamente las diferentes iteraciones que se realizan para finalizar el algoritmo.

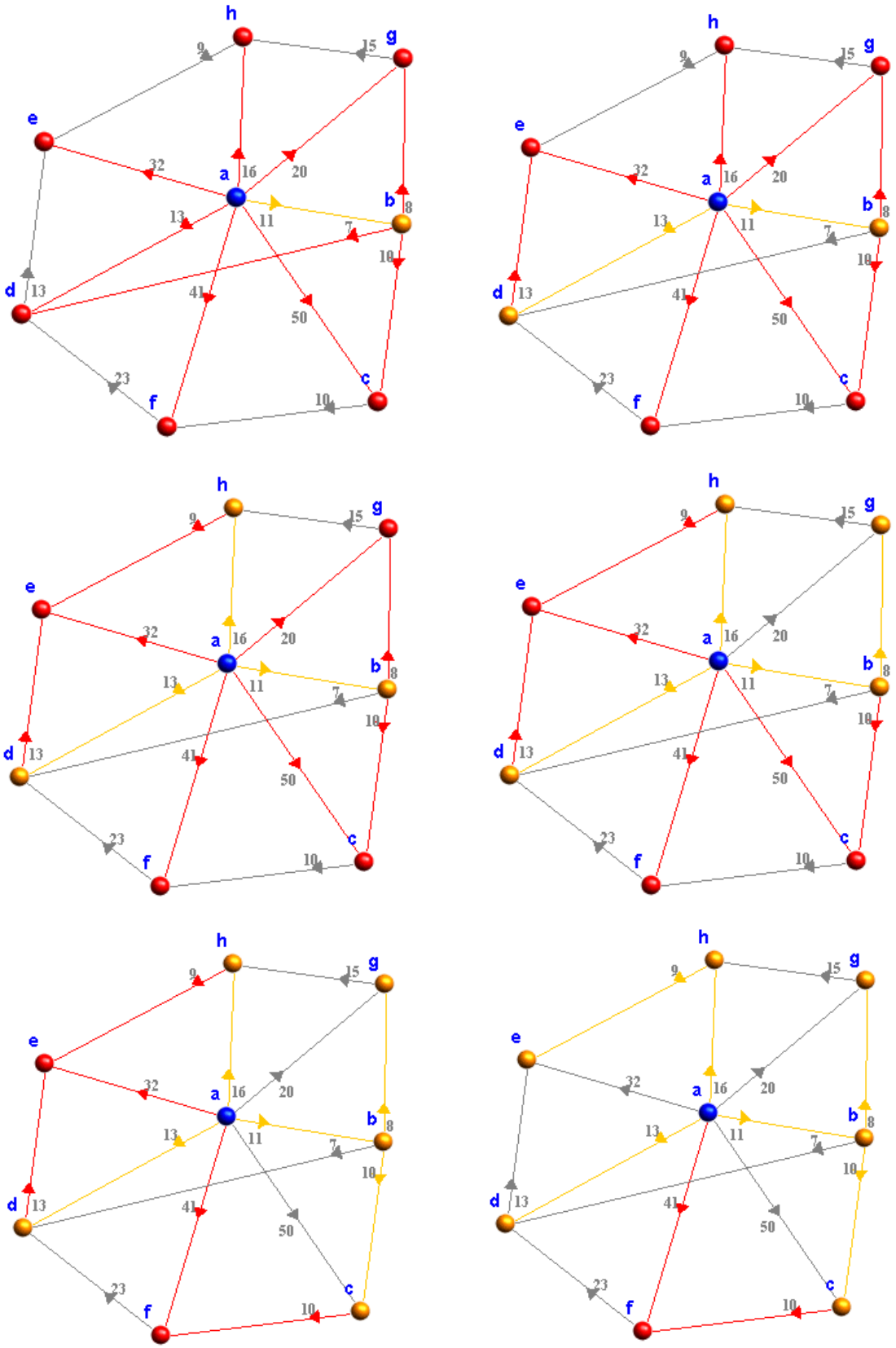


Figura 3.3. Muestra de las 6 primeras iteraciones del algoritmo

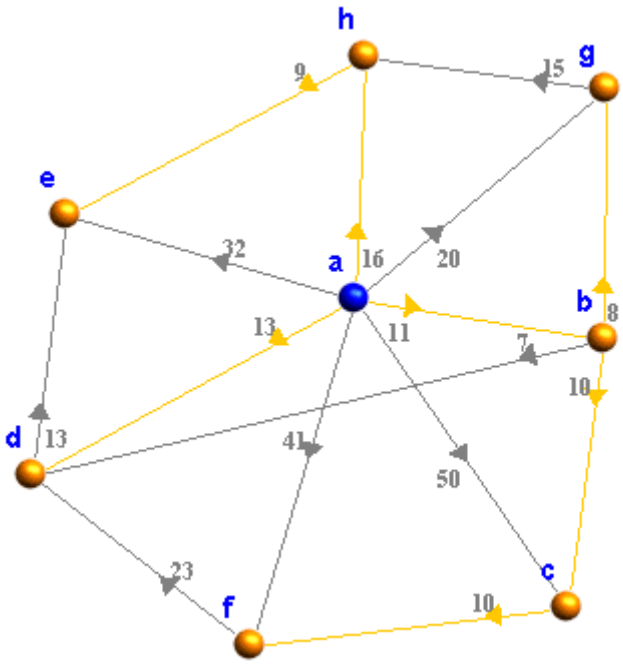


Figura 3.4. Solución final, camino mínimo.[1]

[1] Todas las imágenes de este apartado se han obtenido del la página web http://neo.lcc.uma.es/evirtual/cdd/applets/distancia%20corta/Example2.html

3.3.2. Algoritmo de Floyd-Warshall

Este algoritmo se usa para calcular el camino mínimo en un grafo o digrafo G, pero sólo si las aristas del grafo tienen peso 1. Existe un algoritmo similar para calcular el camino mínimo en un digrafo cuando éste tiene peso. El tiempo de ejecución de este algoritmo es O (n^3).

Para trabajar con grafos en un ordenador utilizamos la matriz de adyacencia comentada en el punto 2.1.3.1., matriz cuadrada de orden n que tiene en el lugar (i,j) el valor del coste del arco (i,j).

Sea $G = (V,E)$ un grafo dirigido con n nodos guardado en forma de matriz de adyacencia $A(i,j)$, pero las posiciones (i,j) correspondientes a los arcos no existentes son iguales a ∞ en lugar de 0. Este algoritmo encuentra la matriz de distancias, D, entre los nodos de dicho grafo cuyas aristas tienen peso de valor 1.

Así, el algoritmo queda formulado:

1. [Inicializar D] repetir para $i,j = 1,2,\dots n$:
   Si $w(i,j) = 0$, entonces: hacer $D(i,j) := \infty$

   Si no: hacer $D(i,j) := w(i,j)$

   [Fin de bucle]

2. [Actualizar D] repetir paso 3 y 4 para $k = 1,2,\dots,n$
3. repetir paso 4 para $i = 1,2,\dots,n$
4. repetir para $j = 1,2,\dots,n$
   hacer $D(i,j) := min\,[D(i,j), D(i,k) + D(k,j)]$.

   [fin de bucle]

   [fin de bucle paso 3]

   [fin de bucle paso 2]

5. Salir

A diferencia del algoritmo de Dijkstra, que considera un nodo origen cada vez, este algoritmo obtiene la ruta más corta entre todo par de nodos (todos con todos). Itera sobre el conjunto de nodos que se permiten como nodos intermedios en los cambios. Empieza con caminos de un solo salto y en cada paso ve si es mejor la ruta de la iteración anterior o si por el contrario conviene ir por otro nodo más.

Lo que se obtiene después de aplicar el algoritmo es la matriz de caminos mínimos, a partir de la cual conocemos la distancia o coste menor entre dos nodos cualesquiera. Lo que no proporciona el algoritmo es el camino a través del cual hemos llegado, pero

podemos conocerlo observando las iteraciones realizadas. Para llegar a obtener la ruta hay que utilizar unas tablas de encaminamiento donde se puede ver la ruta que se ha seguido.

3.3.3. Algoritmo de Bellman-Ford

Este algoritmo halla la mínima distancia desde un nodo dado al resto de nodos. Si se quiere saber el camino por el que se llega con coste o distancia mínima, hay que construir adicionalmente unas tablas de encaminamiento.

Este algoritmo itera sobre el número de saltos, N, que coincide con el diámetro del digrafo, por lo que no se encuentran las rutas óptimas hasta que no se han realizado todas las iteraciones. El vector distancias del grafo se denomina D.

La complejidad del algoritmo es de orden $O(n^3)$ por cada nodo que realiza el algoritmo, es decir, es del mismo orden que el algoritmo de Floyd-Warshall, pero un orden mayor que el algoritmo de Dijkstra.

Así, el algoritmo se formula:

1. [Inicializar D] caminos de i = 0 saltos:

   $D_j^{(o)} = \infty$, para todo $j \neq 1$;

   $D_1 = 0$

2. [Actualizar D] hacer para $i = 1, \ldots n$

   $D_j(i+1) = \min\{D_j(I), D_k(I) + d_{kj}\}$ para todo $I \neq 1$

   [Fin de bucle del paso 2]

3. Salir.

Pero este algoritmo presenta algunos inconvenientes debido a que los vectores de distancia son proporcionales al tamaño de la red. Por esta causa no tienen un buen comportamiento con redes grandes.

Por otro lado, los nodos no conocen la topología de la red, por lo que pueden aparecer bucles transitorios, hasta que los nodos conozcan exactamente la situación de los otros nodos.

3.3.4. Algoritmo de Chen-Yang

El objetivo de este algoritmo es buscar el camino de mínimo coste en una ciudad moderna que tiene controles semafóricos en las intersecciones de las calles.

La ciudad la representamos como una red de transporte $R = (V, E, t, s, d)$, donde V representa el conjunto de las intersecciones, E las calles de la ciudad, t es una función sobre los arcos de manera que $t(u, v)$ el tiempo / coste de ir del nodo u al nodo v, s el nodo de salida u origen y d el nodo de llegada o destino.

El objetivo es encontrar el camino más corto del nodo s al d en R, donde los nodos intermedios están sometidos a un control semafórico, que representaremos mediante ventanas de tiempo.

Como hemos comentado en el tema anterior, desde que podemos representar los semáforos de una manera apropiada, las redes de semáforos han adquirido más importancia. En una intersección, el control semafórico tiene una secuencia repetida de ventanas, de manera que en cada ventana sólo se permiten algunas rutas, y se prohíben otras.

Se descompone cada ruta posible a través de u en un arco entrante (x, u) y un arco saliente (u, y). Así todas las rutas posibles (x, u, y), si quedan escritas <x, u, y> están contenidas en un conjunto de nodos triples correspondientes a una ventana de tiempo. Procediendo así construiremos la red de semáforos que modela la ciudad.

A cada nodo u le asociamos una lista de r ventanas de tiempo $WL(u) = (w_{su}, w_{u,1}, \dots w_{y,r})$ donde w_{su} es el tiempo de salida de la primera ventana y $w_{u,i}$ la i-ésima ventana de tiempo del nodo u, para $i = 1, \dots r$.

Los ciclos semafóricos son repetitivos, por lo que estas ventanas forman una secuencia repetitiva. Es decir,

$$w_{u,i} = w_{u,\lambda r+i} \quad 1 \le i \le r-1, \lambda \in N$$
$$w_{u,r} = w_{u,\lambda r} \quad \lambda \in N$$

O en otras palabras $w_{u,i} = w_{u,j}$ siempre que $i = j$. La secuencia de las ventanas describe una fase completa de la señal.

Para cada ventana $w_{u,i}$ del nodo u, usaremos $d_{u,i}$ para definir su duración. También asociamos un conjunto de nodos triples $NT_{u,i}$ con cada ventana $w_{u,i}$. Un nodo triple <x, u, y> en $NT_{u,i}$ implica que la i-ésima ventana del nodo u permite visitar el nodo y a través del nodo x, es decir, da el conjunto de rutas permitidas en la i-ésima ventana de tiempo del nodo u.

Las etiquetas y funciones que tiene el algoritmo son las siguientes:

- $Arrived(v, u)$, para cada arco (v, u) en A, representa el primer tiempo en el que podemos llegar al nodo u a través del arco (v, u).
- $Leaving\ (v, u, w)$, es el primer tiempo en el que podemos abandonar el nodo u si el arco precedente es (v, u) y el siguiente arco es el arco (u, w).

- $Pred\ (u,w)\ =\ (v,u)$ indica que (v,u) es el arco que precede al arco (u,w) en el camino más corto hasta w.
- $Earliest\ (v,u,w,t)$, computa el tiempo inicial para salir del nodo u hacia el nodo w, a través del arco (u,w) teniendo en cuenta que viniendo del nodo v se llega al nodo u en un tiempo t.

Debido a la existencia de ventanas de tiempo, puede ser posible que no podamos salir de un nodo en el momento en que llegamos, ya que debemos esperar al inicio de la ventana de tiempo correspondiente. De esta manera tenemos la relación

$$Leaving\ (v,u,w)\ =\ Earliest\ (v,u,w,Arrived\ (v,u))$$

Después de salir del nodo u, visitamos el nodo w en el tiempo $Leaving\ (v,u,w)\ +\ t(u,w)$. Como visitamos el nodo u a través de una variedad de arcos (v, u), tenemos

$$Arrived\ (u,w)\ =\ \min_{\forall v}\{Leaving(v,u,w)+t(u,w)\}$$

Así, el algoritmo de Chen y Yang queda formulado:

1. $Arrived(0,s)=0$

   $Arrived(v,u)=\infty$ para todos los arcos (v, u) de A

2. Hallar y cambiar el mínimo elemento de $Arrived\ (v,u)$

3. Si $u=d$ ir al paso 5

4. Para cada arco (u, w) saliente del nodo u, hacer

   Empezar

   $Leaving(v,u,w)=Earliest(v,u,wArrived(v,u))$

   $Temp(u,w)=Leaving(v,u,w)+t(u,w)$

   Si $Temp(u,w)\langle Arrived(u,w)$ luego

   $Arrived(u,w)=Temp(u,w),\Pr ed(u,w)=(v,u)$, y

   Actualizar el valor *Arrived* (v, u)

   Fin

   Ir al paso 2

5. De $Pred\ (v,u)$ hallamos el camino más corto

   Salida $Arrived\ (v,u)$ como el tiempo mínimo.

Ejemplo 3.1.

En la figura 3.5 tenemos una red semafórica con ventanas de tiempo en cada uno de sus nodos, excepto en el nodo de salida $s = 1$ y en el de llegada $d = 6$. Para hallar el coste mínimo entre los nodos s y d se usará el algoritmo de Chen y Yang.

Los números situados en los arcos representan el coste de ir de un nodo a otro. En esta figura también se muestran las restricciones de las ventanas en cada intersección, es decir, los giros permitidos en cada momento. Recordemos que la representación de las ventanas es cíclica, es una secuencia repetida.

Por ejemplo la primera ventana del nodo 2 comienza en el tiempo 0, la ventana w_{1+2i} tiene una duración o coste de 4 y la ventana w_{2+2i} tiene una duración o coste 3, siendo i un número natural.

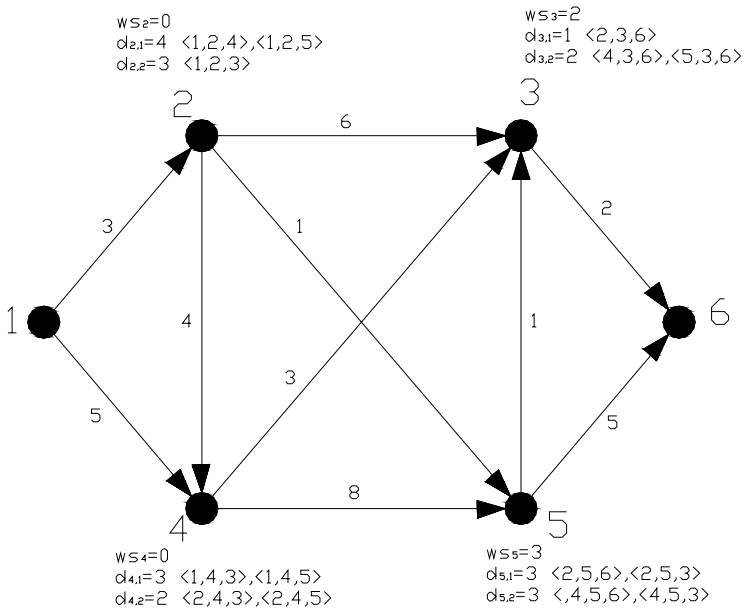


Figura 3.5. Red semafórica

El algoritmo se inicia con $Arrived\ (0,1) = 0$. Del nodo 1 salen dos arcos (1,2) y (1,4). Hallamos el valor de $Arrived$ de cada una de las rutas $Arrived\ (1,2) = 3$ y $Arrived\ (1,4) = 5$. Veamos la siguiente figura. El número entre paréntesis indica el valor de $Arrived\ (i,j)$, donde i es el nodo del que partimos y j el nodo al que llegamos.

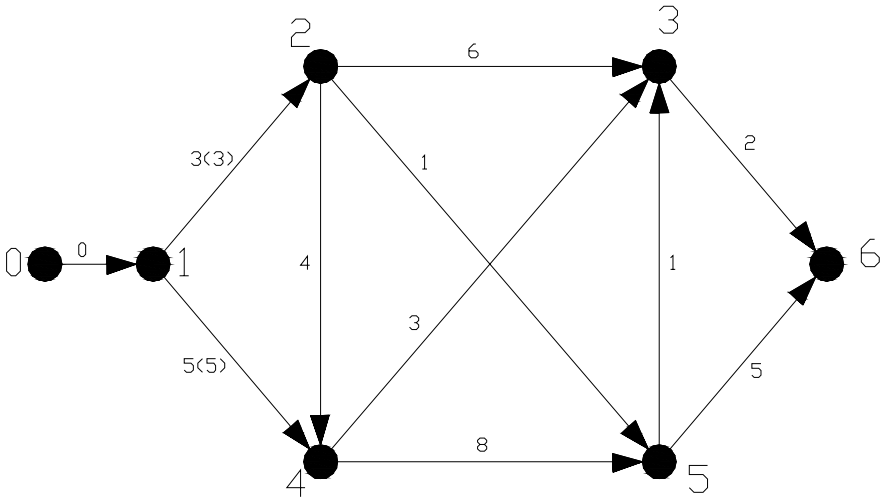


Figura 3.6. Primera iteración

En la segunda iteración se parte de un valor mínimo de $Arrived(1,4) = 3$. La llegada al nodo 4 se produce en el tiempo 3, es decir en la primera ventana, por lo que en ese momento se puede salir hacia los nodos 4 y 5, así $Arrived(2,4) = 7$ y $Arrived(2,5) = 4$. Por el contrario no está permitido salir hacia el nodo 3 hasta el tiempo 4, y esto justifica que $Arrived(2,3) = 10$. El resultado de esta segunda iteración lo vemos en la figura 3.7.

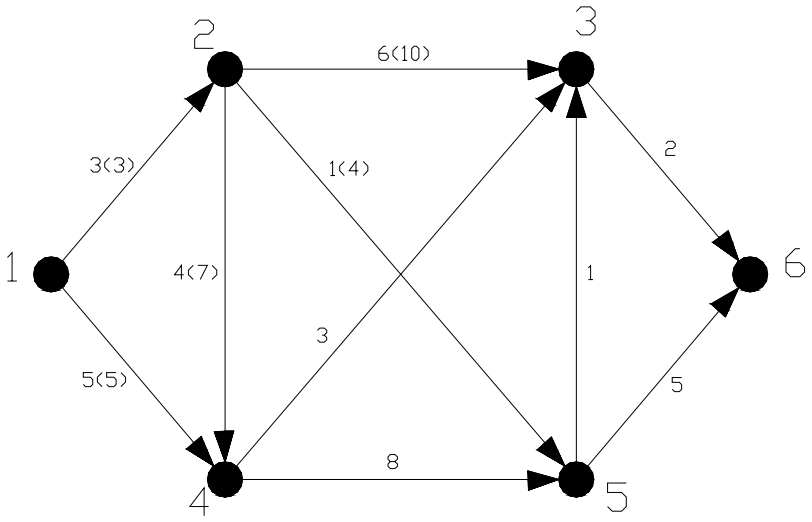


Figura 3.7. Segunda iteración

El mínimo siguiente a considerar es $Arrived(3,5) = 4$. Esta llegada se produce en la ventana 1, así que podemos salir inmediatamente a los nodos 6 y 3, ya que venimos del nodo 2. $Arrived(5,3) = 5, Arrived(5,6) = 9$.

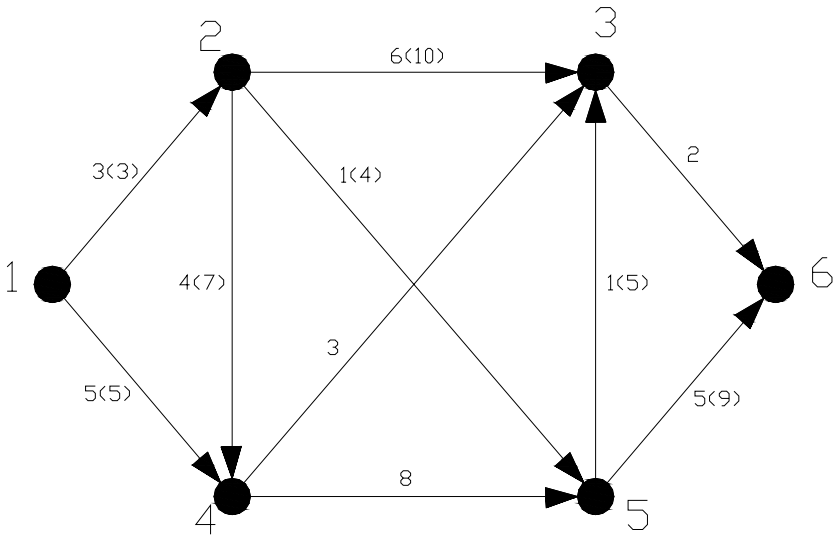


Figura 3.8. Tercera iteración

El siguiente mínimo que consideramos es $Arrived(1,4) = 5$. No hay penalización, ya que la llegada se produce en la ventana de tiempo que nos permite salir inmediatamente hacia los nodos 3 y 5. $Arrived(4,3) = 8$ y $Arrived(4,5) = 13$.

Para la quinta iteración partimos del mínimo $Arrived(5,3) = 5$. Al llegar fuera de la ventana de tiempo que permite salir hacia el nodo de llegada existe la penalización de coste 1, ya que la ventana se abre en el tiempo 6. $Arrived(3,6) = 8$.

Así en la sexta iteración se cumple la condición que finaliza el algoritmo. Así el camino mínimo es (1,2,5,3,6) y tiene un coste de 8.

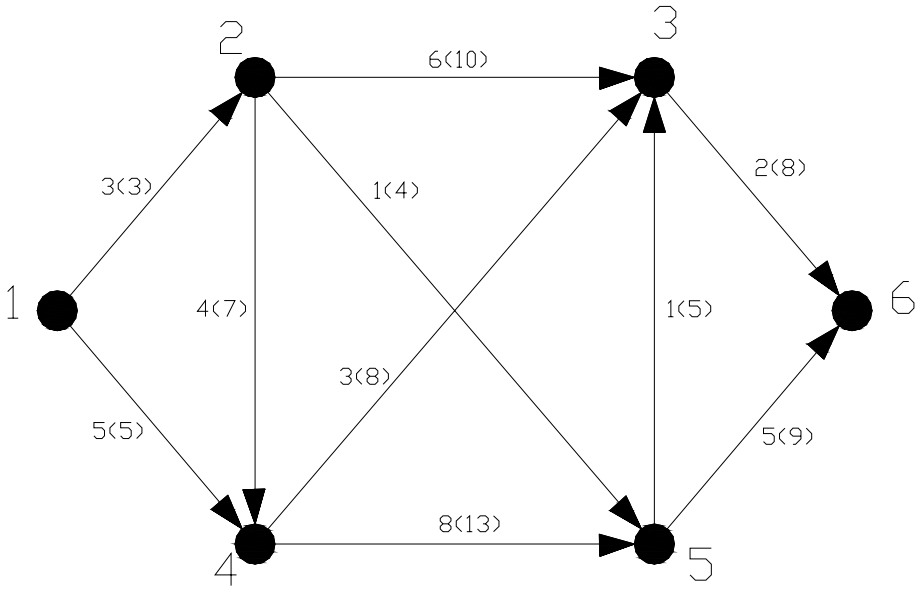


Figura 3.9. Cuarta a última iteración

4. Algoritmo de Ford y Fulkerson

El algoritmo de Ford y Fulkerson se basa en el teorema de flujo máximo – corte mínimo para encontrar el flujo máximo en una red.

Este método, probado por Ford-Fulkerson en 1956, depende de dos conceptos importantes: *red residual* y *camino de aumento*. Este método procede iterativamente. Comienza con $f(u,v) = 0$ para todo $u, v \in V$ con lo que el flujo inicial vale 0. En cada iteración se incrementa el valor del flujo buscando un camino de aumento, que puede interpretarse como un camino de s a t por el cual se puede enviar más flujo y por tanto aumentar el flujo a través de este camino. Este proceso se repite hasta que no existan más caminos de aumento.

Intuitivamente, dada una red de transporte y un flujo, la *red residual* está formada por aristas que admiten más flujo. Más formalmente, supongamos que tenemos una red de transporte $G = (V,E)$ con una fuente s y un sumidero t. Sea f un flujo en G y sean un par de vértices $u, v \in V$. La cantidad de flujo adicional que podemos enviar de u a v sin exceder la capacidad $c_f(u,v)$ es la *capacidad residual* de u a v expresada como: $c_f(u,v) = c(u,v) - f(u,v)$.

Dada una red de transporte $G = (V,E)$ y un flujo f, la red residual de G inducida por f es $G_f = (V, E_f)$, donde

$$E_f = \{(u,v) \in VxV : c_f(u,v) > 0\}$$

Las aristas de E_f son aristas de E o sus traspuestas. Si $f(u,v) < c(u,v)$ para una arista (u,v), entonces $c_f(u,v) = c(u,v) - f(u,v) > 0$ y por tanto $(u,v) \in E_f$. Si $f(u,v) > 0$ para una arista (u,v), entonces $f(v,u) < 0$. En tal caso $c_f(v,u) = c(v,u) - f(v,u) > 0$, y por tanto $(v,u) \in E_f$. Nótese que una arista (u,v) aparece en la red residual si (u,v) o (v,u) están en la red original.

Dada una red de transporte $G = (V,E)$ y un flujo f, un *camino de aumento* o ruta de penetración Np es un camino simple de s a t en la red residual G_f. Cada arista (u,v) en un camino de aumento admite un flujo adicional positivo de u a v sin violar la restricción de capacidad de la arista.

Se define la *capacidad residual* de un camino de aumento p como:

$$c_f(p) = \min \{c_f(u,v) : (u,v)p \}$$

Dada una red de transporte $G = (V, E)$ con una fuente s y un sumidero t y un flujo f, se puede demostrar que f es un flujo máximo en G si y solo si la red residual G_f no contiene ningún camino de aumento.

Con las definiciones anteriores podemos introducir el método de Ford-Fulkerson para computar el flujo máximo en una red de transporte.

Así, el algoritmo de Ford-Fulkerson queda de la siguiente manera. Empezando con un flujo arbitrario − por ejemplo, el flujo nulo −, sistemáticamente se buscan los caminos en los que existe la posibilidad de aumentar el flujo del origen al destino. Los vértices son etiquetados para indicar a través de que arco, y de cuánto, se incrementa el flujo. Cuando se encuentra una secuencia de aumento de flujo, incrementamos el flujo a su máxima capacidad a lo largo del camino y todas las etiquetas son borradas. El algoritmo acaba cuando el flujo ya no se puede aumentar más.

Así, tenemos:

Algoritmo. Sea C_{ij} y C_{ji} las capacidades iniciales del arco que une el nodo i y el nodo j, y c_{ij} y c_{ji} las capacidades residuales, para un nodo j que recibe el flujo del nodo i, definimos una clasificación $[f_j, i]$, donde f_j es el flujo del nodo i al nodo j.

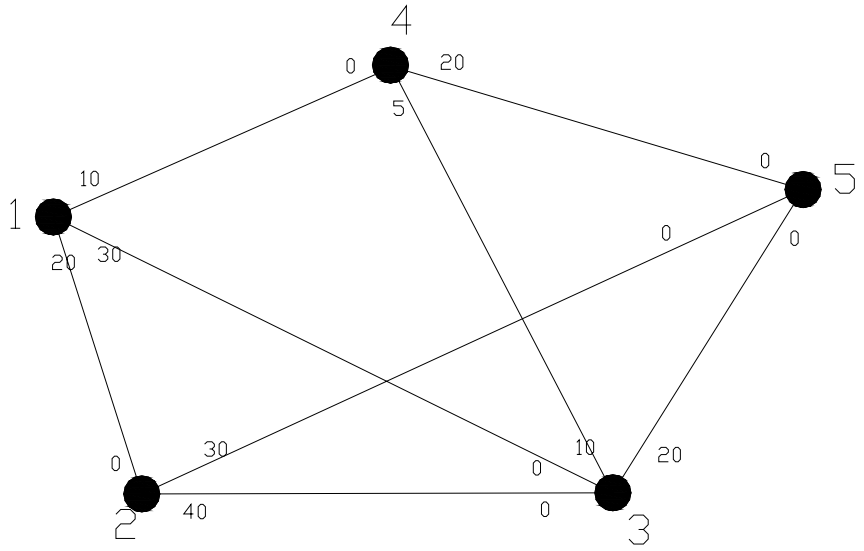
- ***Paso 1***: Inicializamos las capacidades residuales a las capacidades iniciales, hacemos $(c_{ij}, c_{ji}) = (C_{ij}, C_{ji})$ para todo arco de la red. Suponiendo el nodo 1 como el nodo origen, hacemos $f_1 = \infty$ y clasificamos el nodo origen con ***[∞,-]***. Tomamos *i=1* y vamos al paso 2.
- ***Paso 2:*** Determinamos S_i como un conjunto que contendrá los nodos a los que podemos acceder directamente desde *i* por medio de un arco con capacidad positiva, y que no formen parte del camino en curso. Si S_i contiene algún nodo vamos al paso 3, en el caso de que el conjunto sea vacío saltamos al paso 4.
- ***Paso 3:*** Obtenemos $k \in S_i$ como el nodo destino del arco de mayor capacidad que salga de *i* hacia un nodo perteneciente a S_i. Es decir, $c_{ik} = max\{c_{ij}\}$ ***con*** $j \in S_i$. Hacemos $f_k = c_{ik}$ y clasificamos el nodo *k* con $[f_k, i]$. Si *k* es igual al nodo destino o sumidero, entonces hemos encontrado una ruta de penetración, vamos al paso 5. En caso contrario continuamos con el camino, hacemos *i=k* y volvemos al paso 2.
- ***Paso 4 (retroceso):*** Si *i=1*, estamos en el nodo origen, y como S_i es vacío, entonces no podemos acceder a ningún nodo, ni encontrar algún nuevo camino, hemos terminado, vamos al paso 6. En caso contrario, $i \neq 1$, le damos al valor *i* el del nodo que se ha clasificado inmediatamente antes, eliminamos *i* del conjunto S_i actual y volvemos al paso 2.
- ***Paso 5:*** Llegados a este paso tenemos un nuevo camino: $Np = \{1, k_1, k_2, \dots, n\}$, esta será la *p-ésima* ruta de penetración desde el nodo origen al nodo destino. El flujo máximo a lo largo de esta ruta será la capacidad mínima de las capacidades residuales de los arcos que forman el camino, es decir: $fp = min\{a_1, a_{k1}, a_{k2}, \dots, a_n\}$.
  La capacidad residual de cada arco a lo largo de la ruta de penetración se disminuye por f_p en dirección del flujo y se incrementa por f_p en dirección

inversa, es decir, para los nodos *i* y *j* en la ruta, el flujo residual se cambia de la (c_{ij}, c_{ji}) actual a $(c_{ij} - f_p, c_{ji} + f_p)$ si el flujo es de *i* a *j*, o $\left(c_{ij} + f_p, c_{ji} - f_p\right)$ si el flujo es de *j* a *i*. Inicializamos *i=1* y volvemos al paso 2 para intentar una nueva ruta de penetración.

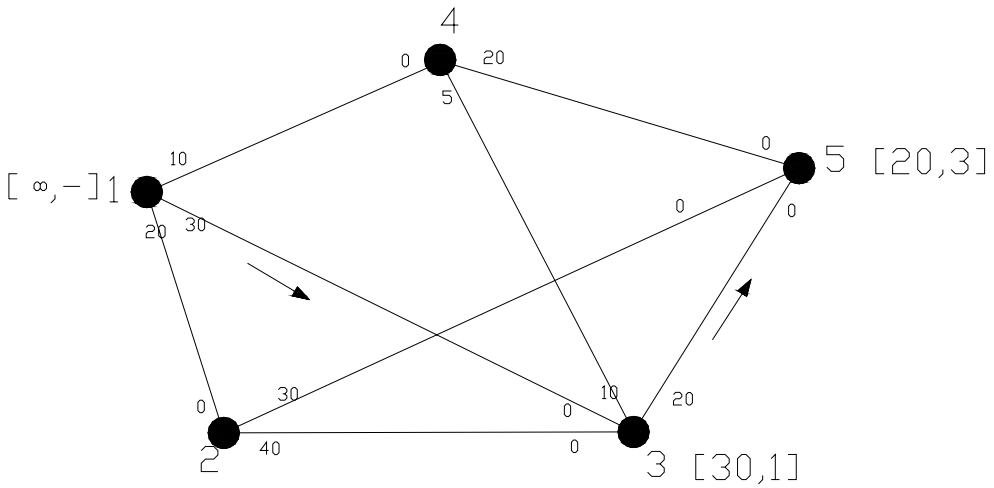
- ***Paso 6 (Solución):*** Una vez aquí, hemos determinado *m* rutas de penetración. El flujo máximo en la red será la suma de los flujos máximos en cada ruta obtenida, es decir: $F = f_1 + f_2 + \ldots + f_m$. Teniendo en cuenta que las capacidades residuales inicial y final del arco *(i, j)* las dan $\left(C_{ij}, C_{ji}\right)$ y (c_{ij}, c_{ji}) respectivamente, el flujo máximo para cada arco se calcula como sigue: sea $(\alpha, \beta) = (C_{ij} - c_{ij}, C_{ji} - c_{ji})$, si *α>0*, el flujo óptimo de *i* a *j* es *α*, de lo contrario, si *β>0*, el flujo óptimo de *j* a *i* es *β*. Es imposible lograr que tanto *α* como *β* sean positivas.

Vemos a continuación un ejemplo de aplicación del algoritmo.

Ejemplo 4.1.[2] Queremos determinar el flujo máximo en la red siguiente:



- ***Iteración 1:***



Determinamos las residuales finales (c_{ij}, c_{ji}) iguales a las capacidades iniciales (C_{ij}, C_{ji}).

- **Paso 1:** Hacemos $f_i = \infty$, y clasificamos el nodo 1 con $[f_1, -]$. Tomamos *i=1*.
- **Paso 2:** $S_1 = \{2,3,4\}$ (no vacío).
- **Paso 3:** *k=3* ya que $c_{13} = max\{c_{12}, c_{13}, c_{14}\} = \{20,30,10\} = 30$. Hacemos $f_3 = c_{13} = 30$ y clasificamos el nodo 3 con *[30,1]*. Tomamos *i=3* y repetimos el paso 2.
- **Paso 2:** $S_3 = \{4,5\}$
- **Paso 3:** *k=5* y $f_5 = c_{35} = max\{10,20\} = 20$. Clasificamos el nodo 5 con *[20,3]*. Logramos la penetración, vamos al paso 5.

[2] Este ejemplo lo encontramos en la página web:
http://lear.inforg.uniovi.es/ioperativa/TutorialGrafos/flujomax/flujomax.htm

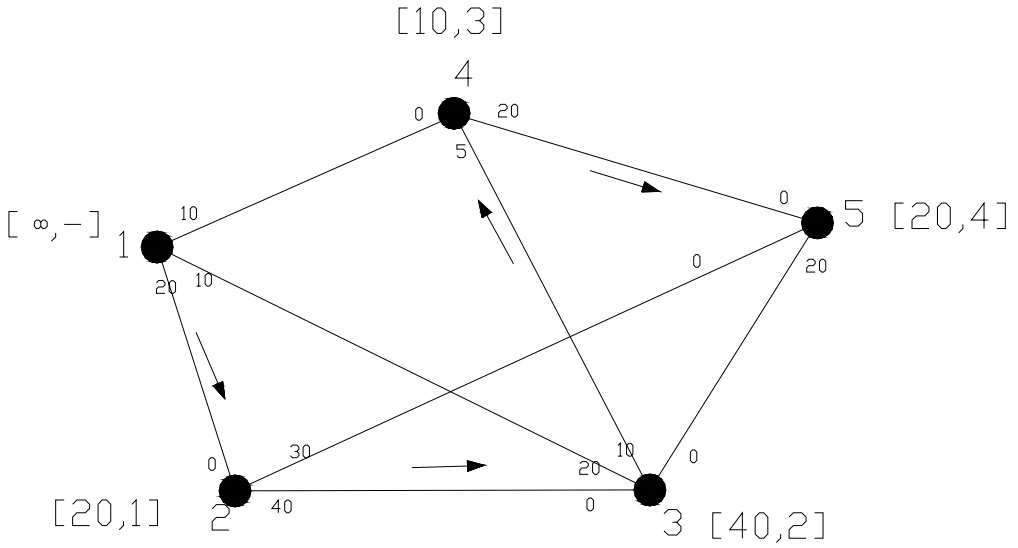
- **Paso 5:** La ruta de la penetración se determina de las clasificaciones empezando en el nodo 5 y terminando en el nodo 1, es decir, 5→*[20,3]*→3→*[30,1]*→1.

Entonces la ruta es *N1={1,3,5}* y $F_1 = min\{f_1, f_3, f_5\} = \{\infty, 30, 20\} = \mathbf{20}$. Las capacidades residuales a lo largo de esta ruta son:

$$(\boldsymbol{c_{13}}, \boldsymbol{c_{31}}) = (30 - 20, 0 + 20) = (\mathbf{10}, \mathbf{20})$$

$$(\boldsymbol{c_{35}}, \boldsymbol{c_{53}}) = (20 - 20, 0 + 20) = (\mathbf{0}, \mathbf{20})$$

- ***Iteración 2:***



- **Paso 1:** Hacemos $f_i = \infty$, y clasificamos el nodo 1 con $[f_1, -]$. Tomamos *i=1*.
- **Paso 2:** $S_1 = \{2,3,4\}$.
- **Paso 3:** *k=2* y $f_2 = c_{12} = max\{20,10,10\} = 20$. Clasificamos el nodo 2 con *[20,1]*. Tomamos *i=2* y repetimos el paso 2.
- **Paso 2:** $S_2 = \{3,5\}$
- **Paso 3:** *k=3* y $f_3 = c_{23} = max\{30,40\} = 40$. Clasificamos el nodo 3 con *[40,2]*. Tomamos *i=3* y repetimos el paso 2.
- **Paso 2:** $S_3 = \{4\}$ $(c_{35} = 0)$, el nodo 1 ya ha sido clasificado y el nodo 2 cumple ambas condiciones, por tanto los nodos 1, 2 y 5 no pueden ser incluidos en S_3).
- **Paso 3:** *k=4* y $f_4 = c_{34} = 10$. Clasificamos el nodo 4 con *[10,3]*. Tomamos *i=4* y repetimos el paso 2.
- **Paso 2:** $S_4 = \{5\}$
- **Paso 3:** *k=5* y $f_5 = c_{45} = 20$. Clasificamos el nodo 5 con *[20,4]*. Logramos la penetración, vamos al paso 5.
- **Paso 5:** La ruta de la penetración es:
  5→*[20,4]*→4→*[10,3]*→3→*[40,2]*→2→*[20,1]*→1.

Entonces la ruta es *N2={1,2,3,4,5}* y $F_2 = min\{\infty, 20{,}40{,}10{,}20\} = \mathbf{10}$. Las capacidades residuales a lo largo de esta ruta son:

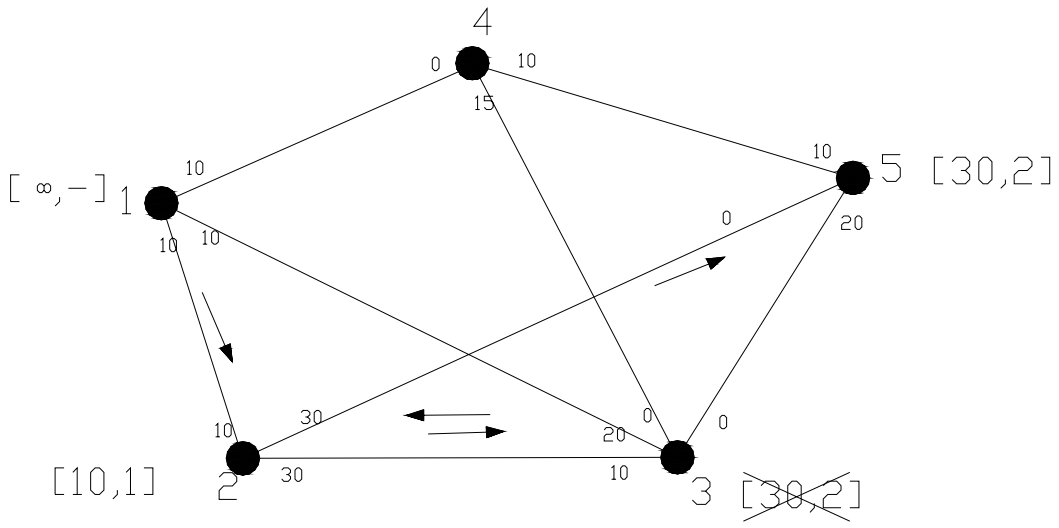
$$(\boldsymbol{c_{12}}, \boldsymbol{c_{21}}) = (20 - 10, 0 + 10) = (\mathbf{10}, \mathbf{10})$$

$$(\boldsymbol{c_{23}}, \boldsymbol{c_{32}}) = (40 - 10, 0 + 10) = (\mathbf{30}, \mathbf{10})$$

$$(\boldsymbol{c_{34}}, \boldsymbol{c_{43}}) = (10 - 10, 5 + 10) = (\mathbf{0}, \mathbf{15})$$

$$(\boldsymbol{c_{45}}, \boldsymbol{c_{54}}) = (20 - 10, 0 + 10) = (\mathbf{10}, \mathbf{10})$$

- ***Iteración 3:***



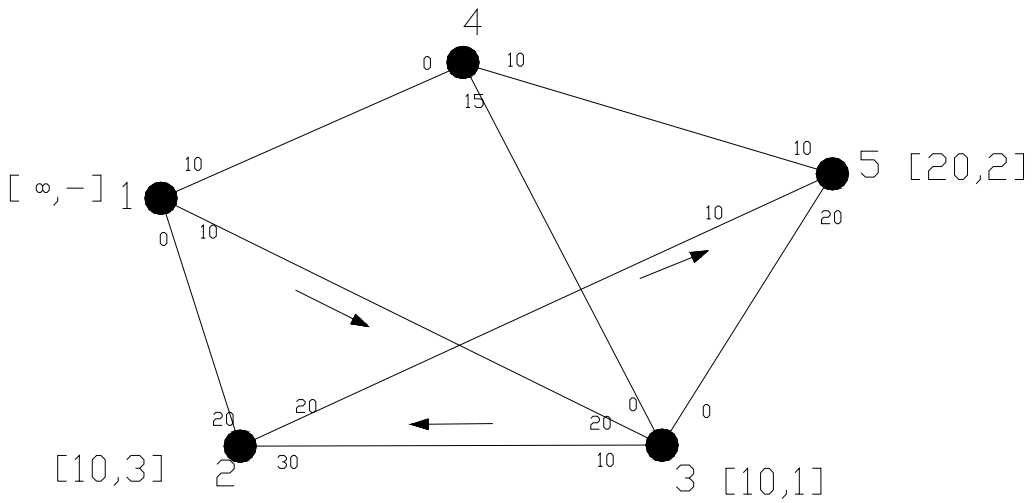
- **Paso 1:** Hacemos $f_i = \infty$, y clasificamos el nodo 1 con *[a1,-]*. Tomamos *i=1*.
- **Paso 2:** $S_1 = \{2{,}3{,}4\}$.
- **Paso 3:** *k=2* y $f_2 = c_{12} = max\{10{,}10{,}10\} = 10$, rompemos el empate arbitrariamente. Clasificamos el nodo 2 con *[10,1]*. Tomamos *i=2* y repetimos el paso 2.
- **Paso 2:** $S_2 = \{3{,}5\}$
- **Paso 3:** *k=3* y $f_3 = c_{23} = max\{30{,}30\} = 30$. Clasificamos el nodo 3 con *[30,2]*. Tomamos *i=3* y repetimos el paso 2.
- **Paso 2:** S_3 vacío ya que $c_{34} = c_{35} = 0$. Vamos al paso 4 para retroceder.
- **Paso 4:** La clasificación *[30,2]* nos dice que el nodo inmediatamente precedente es el 2. Eliminamos el nodo 3 de una consideración posterior en esta iteración. Tomamos *i=2* y repetimos el paso 2.
- **Paso 2:** $S_2 = \{5\}$
- **Paso 3:** *k=5* y $f_5 = c_{25} = 30$. Clasificamos el nodo 5 con *[30,2]*. Logramos la penetración, vamos al paso 5.
- **Paso 5:** La ruta de la penetración es: 5→*[30,2]*→2→*[10,1]*→1.

Entonces la ruta es *N2={1,2,5}* y $F_3 = min\{\infty, 10,30\} = \mathbf{10}$. Las capacidades residuales a lo largo de esta ruta son:

$$(\boldsymbol{c_{12}}, \boldsymbol{c_{21}}) = (10 - 10, 10 + 10) = (\mathbf{0}, \mathbf{20})$$

$$(\boldsymbol{c_{25}}, \boldsymbol{c_{52}}) = (30 - 10, 0 + 10) = (\mathbf{20}, \mathbf{10})$$

- ***Iteración 4:***



- **Paso 1:** Hacemos $f_i = \infty$, y clasificamos el nodo 1 con *[a1,-]*. Tomamos *i=1*.
- **Paso 2:** $S_1 = \{3,4\}$.
- **Paso 3:** *k=3* y $f_3 = c_{13} = max\{10,10\} = 10$. Clasificamos el nodo 3 con *[10,1]*. Tomamos *i=3* y repetimos el paso 2.
- **Paso 2:** $S_3 = \{2\}$
- **Paso 3:** *k=2* y $f_2 = c_{32} = 10$. Clasificamos el nodo 2 con *[10,3]*. Tomamos *i=2* y repetimos el paso 2.
- **Paso 2:** $S_3 = \{5\}$
- **Paso 3:** *k=5* y $f_5 = c_{25} = 20$. Clasificamos el nodo 5 con *[20,2]*. Logramos la penetración, vamos al paso 5.
- **Paso 5:** La ruta de la penetración es: 5→*[20,2]*→2→*[10,3]*→3→*[10,1]*→1.

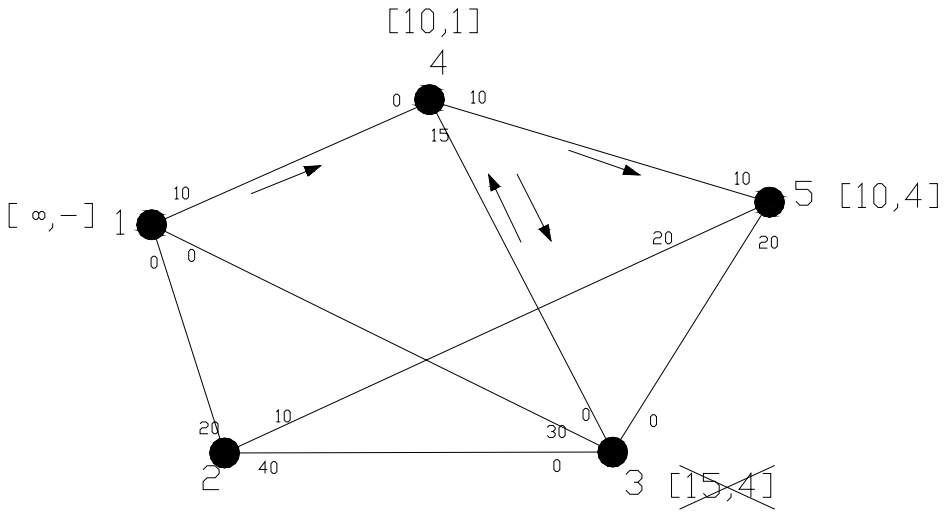
Entonces la ruta es *N4={1,3,2,5}* y $F_4 = min\{\infty, 10,10,20\} = \mathbf{10}$. Las capacidades residuales a lo largo de esta ruta son:

$$(\boldsymbol{c_{13}}, \boldsymbol{c_{31}}) = (10 - 10, 20 + 10) = (\mathbf{0}, \mathbf{30})$$

$$(\boldsymbol{c_{32}}, \boldsymbol{c_{23}}) = (10 - 10, 30 + 10) = (\mathbf{0}, \mathbf{40})$$

$$(\boldsymbol{c_{25}}, \boldsymbol{c_{52}}) = (20 - 10, 10 + 10) = (\mathbf{10}, \mathbf{20})$$

- ***Iteración 5:***



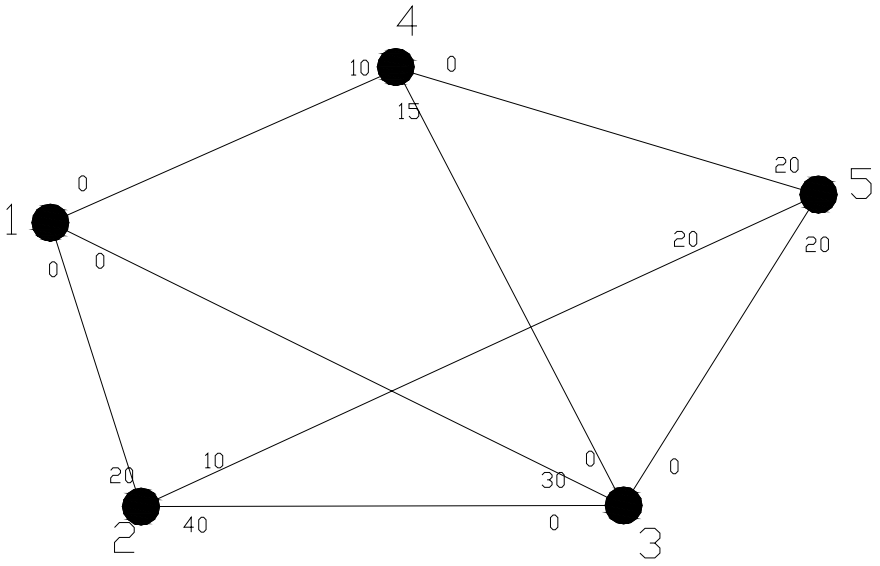
- **Paso 1:** Hacemos $f_i = \infty$, y clasificamos el nodo 1 con *[a1,-]*. Tomamos *i=1*.
- **Paso 2:** $S_1 = \{4\}$.
- **Paso 3:** *k=4* y $f_4 = c_{14} = 10$. Clasificamos el nodo 4 con *[10,1]*. Tomamos *i=4* y repetimos el paso 2.
- **Paso 2:**$S_4 = \{3,5\}$
- **Paso 3:** *k=3* y $f_3 = c_{23} = max\{15,10\} = 15$. Clasificamos el nodo 3 con *[15,4]*. Tomamos *i=3* y repetimos el paso 2.
- **Paso 2:** S_3 vacío ya que $c_{32} = c_{34} = c_{35} = 0$. Vamos al paso 4 para retroceder.
- **Paso 4:** La clasificación *[15,4]* nos dice que el nodo inmediatamente precedente es el 4. Eliminamos el nodo 3 de una consideración posterior en esta iteración. Tomamos *i=4* y repetimos el paso 2.
- **Paso 2:** $S_4 = \{5\}$
- **Paso 3:** *k=5* y $f_5 = c_{45} = 10$. Clasificamos el nodo 5 con *[10,4]*. Logramos la penetración, vamos al paso 5.
- **Paso 5:** La ruta de la penetración es: 5→*[10,4]*→4→*[10,1]*→1.

Entonces la ruta es *N2={1,4,5}* y $F_3 = min\{\infty, 10,10\} = \mathbf{10}$. Las capacidades residuales a lo largo de esta ruta son:

$$(\boldsymbol{c_{14}, c_{41}}) = (10 - 10, 0 + 10) = (\mathbf{0}, \mathbf{10})$$

$$(\boldsymbol{c_{45}, c_{54}}) = (10 - 10, 10 + 10) = (\mathbf{0}, \mathbf{20})$$

- ***Iteración 6:***



No son posibles más penetraciones, debido a que todos los arcos fuera del nodo 1 tienen residuales cero. Vayamos al paso 6 para determinar la solución.

- ***Paso 6*:** El flujo máximo en la red es $\boldsymbol{F = f_1 + f_2 + \ldots + f_5 = 60\ unidades}$. El flujo en los diferentes arcos se calcula restando las últimas residuales obtenidas en la última iteración de las capacidades iniciales:

Arco	(C_{ij}, C_{ji}) - (c_{ij}, c_{ji})	Cantidad de flujo	Dirección
(1,2)	(20,0)-(0,20)=(20,-20)	20	1 a 2
(1,3)	(30,0)-(0,30)=(30,-30)	30	1 a 3
(1,4)	(10,0)-(0,10)=(10,-10)	10	1 a 4
(2,3)	(40,0)-(40,0)=(0,0)	0	-
(2,5)	(30,0)-(10,20)=(20,-20)	20	2 a 5
(3,4)	(10,5)-(0,15)=(10,-10)	10	3 a 4
(3,5)	(20,0)-(0,20)=(20,-20)	20	3 a 5
(4,5)	(20,0)-(0,20)=(20,-20)	20	4 a 5

Tabla 3. Flujo en los distintos arcos

5. Creación del programa de rutas de flujo máximo

El problema del camino de flujo máximo en una red de transporte consiste en hallar el camino entre dos nodos que admite un mayor flujo circulando a través suyo. Para analizar y entender este problema estudiaremos con detenimiento el algoritmo de Ford y Fulkerson.

Desde el principio mi intención era analizar distintos ejemplos reales mediante algún programa de cálculo, pero tras el estudio del algoritmo de Ford y Fulkerson, y dado que no conozco ningún programa al menos de carácter gratuito que te permita ejecutar este algoritmo, quiero crear una aplicación para estudiarlo extensamente, y posteriormente, para poderlo aplicar a redes de diferentes tamaños definidas por mí.

Crearé esta aplicación mediante el programa *'Microsoft Visual Studio 2008®'*, concretamente con el lenguaje Visual Basic .NET. A continuación, se explicará cómo funciona la aplicación, y veremos su diagrama de flujo. El código de programación del programa se presentará en el *Anejo 1. Código del programa*.

El esquema del programa realizado es el que se muestra a continuación.

5.1. Presentación del programa

Fase I. Inicialización

Una vez se ha definido el grafo, mediante la inserción de nodos y aristas en pantalla, se genera la matriz $flujo\ (i,j)$, que almacena, si existe, el flujo del nodo i al nodo j.

Entrada de datos, definimos el nodo origen y el nodo destino.

$[\infty, -]$, primera etiqueta para definir camino. Al partir de s, nodo origen, la etiqueta indica que el flujo para llegar a s es infinito y no venimos de ningún nodo.

Fase II. Cuerpo del programa

Paso 1: Buscar la arista con flujo máximo que parte del nodo v en el que estamos. Etiquetar el nodo al que llegamos, u, como $[f, v]$, siendo f el flujo de v a u, y v el nodo del que venimos.

Paso 2: Si $u = d$, donde d es el nodo destino, ir a paso 5.

Paso 3: Repetir paso 1 hasta que $u = d$.

Paso 4: Retrocediendo desde el nodo d, nodo destino, hasta el nodo s, nodo origen, siguiendo las etiquetas $[f, v]$, obtenemos el camino a seguir. El flujo del camino será el $min\{f \setminus f \in [f, v]\}$.

Paso 5: En la matriz $flujo(i, j)$, se suma $(f, -f)$ a todas las artistas del camino, saliendo del nodo origen al nodo destino.

Paso 6: Almacenamos el camino y su flujo f en la matriz $camino(i)$.

Paso 7: Volvemos a nodo s, nodo origen, y vamos a paso 1 hasta que no exista arista con flujo máximo que parta del origen, en ese caso, ir a fase III.

Fase III. Salida de resultados

De camino(i), sacamos el flujo máximo de la red sumando los flujos de todos los caminos.

Salida de cada camino(i) sobresaliendo en color rojo sobre el grafo inicial, con el flujo que le corresponda.

Saluda del flujo máximo de la red.

5.2. Entrada de datos

Cuando abrimos el programa “Algoritmo de Ford y Fulkerson”, aparece la siguiente ventana:

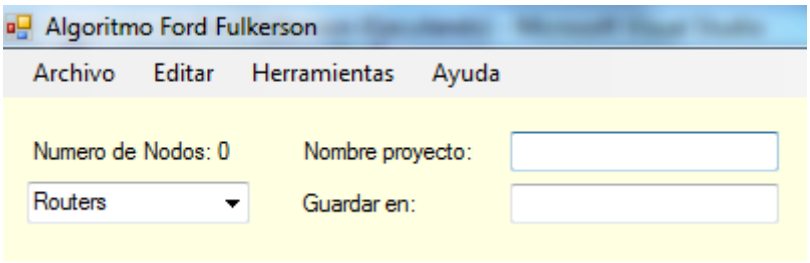


Figura 5.1. Menú principal

Observamos que en el menú de herramientas tenemos distintas opciones y que tenemos una etiqueta que cuenta el número de nodos que estamos usando. Además, vemos un menú desplegable donde se puede leer Routers, y es que podemos elegir si queremos que nuestros nodos sean routers o satélites, que son los dos ejemplos que utilizaremos en este estudio.

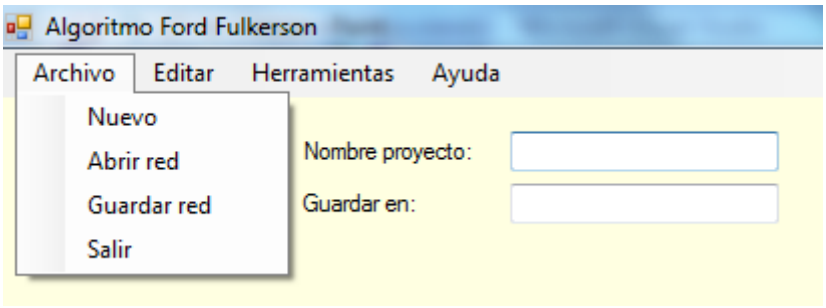


Figura 5.2. Menú archivo

En la figura 5.2. vemos las opciones que aparecen en el menú Archivo. Tenemos la opción Nuevo, que reinicia el programa y limpia todos los resultados que haya en pantalla. Aparece también la opción Abrir red, que carga cualquier red que hayamos definido previamente.

La opción Guardar red crea un archivo con formato .txt que almacena todos los parámetros necesarios para posteriormente poder cargar la red que hayamos definido en el programa. A continuación vemos una figura con un archivo ejemplo:

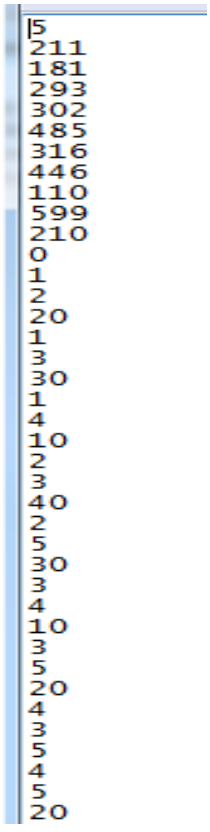


Figura 5.3. Archivo guardado

En este archivo se introduce lo siguiente: en la primera línea se introduce el número de nodos de la red y a continuación las coordenadas de cada nodo (x, y), en orden, del nodo 1 al nodo n. A continuación se introducirá la matriz $flujo(i, j)$, que en se guarda en este archivo de la forma que vemos. Primero introduce el nodo de salida de la arista, después el nodo de llegada y posteriormente el flujo a través de esa arista. En la figura 5.3. la arista $(1,2)$ tiene un flujo $f = 20$, por ejemplo.

En el menú editar podemos insertar enlaces o insertar nodos. Para insertar un enlace debemos insertar previamente como mínimo dos nodos.

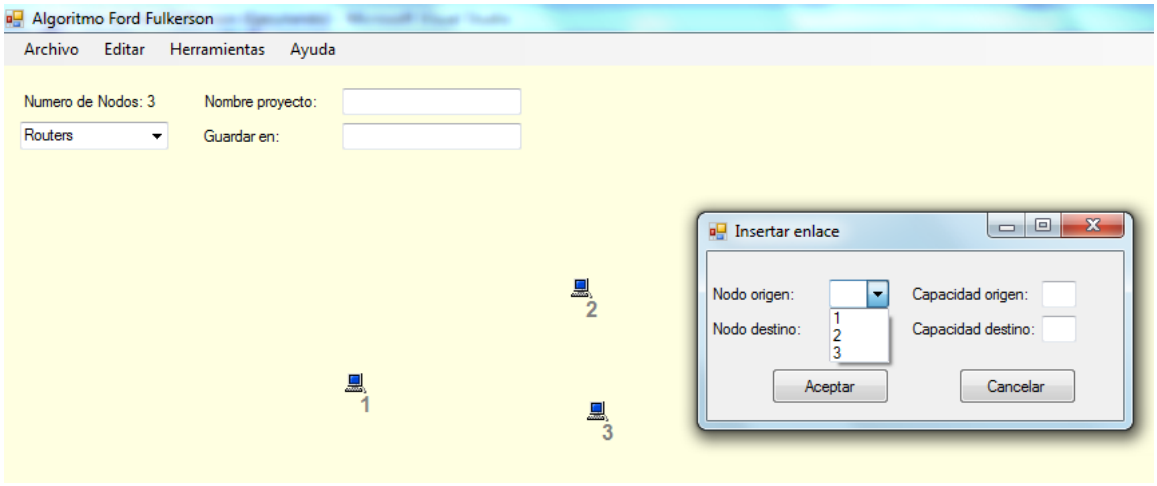


Figura 5.4. Ventana insertar enlace

En la figura 5.4 se ve la ventana que aparece cuando insertamos un enlace. Vemos que debemos introducir el nodo origen y el nodo destino de la arista o enlace a insertar, así como la capacidad desde origen y la capacidad desde destino.

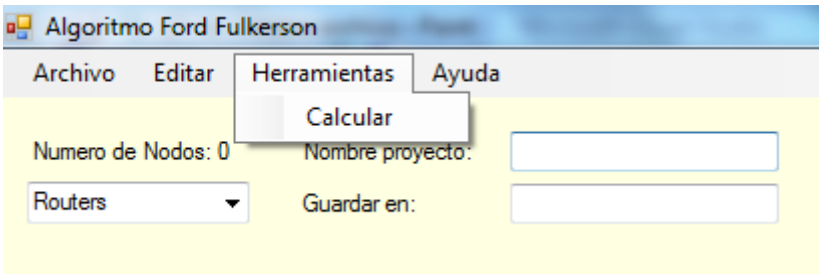


Figura 5.5. Menú Herramientas

Vemos en la figura 5.5. el menú Herramientas del programa, que está compuesto por la opción Calcular, que ejecuta el algoritmo sobre la red que se haya definido previamente.

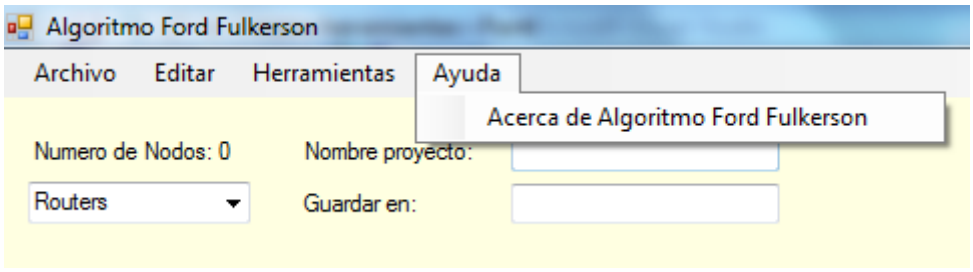


Figura 5.6. Menú Ayuda

Por último, el programa ofrece el menú ayuda (figura 5.6.), que incluye la pestaña Acerca del algoritmo Ford y Fulkerson, donde encontramos alguna limitación del programa y otros datos como el nombre del autor, etc.

5.3. Cuerpo del programa

Una vez definido el grafo y elegida la opción calcular, se ejecuta lo que es el cuerpo del programa, la ejecución del algoritmo propiamente dicha.

Vemos en la figura siguiente un ejemplo de red cargada:

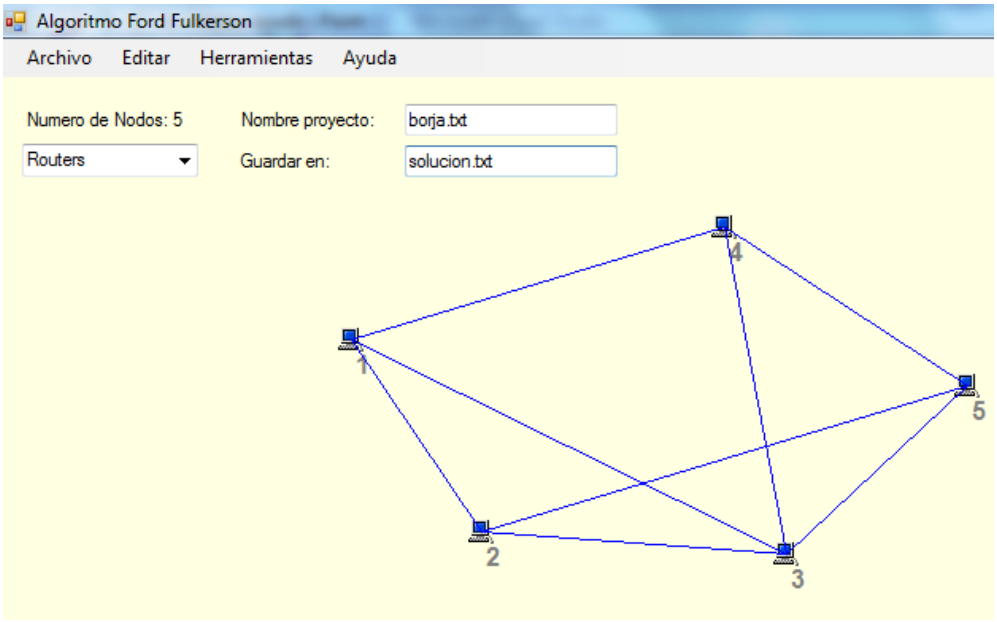


Figura 5.7. Red cargada en el programa

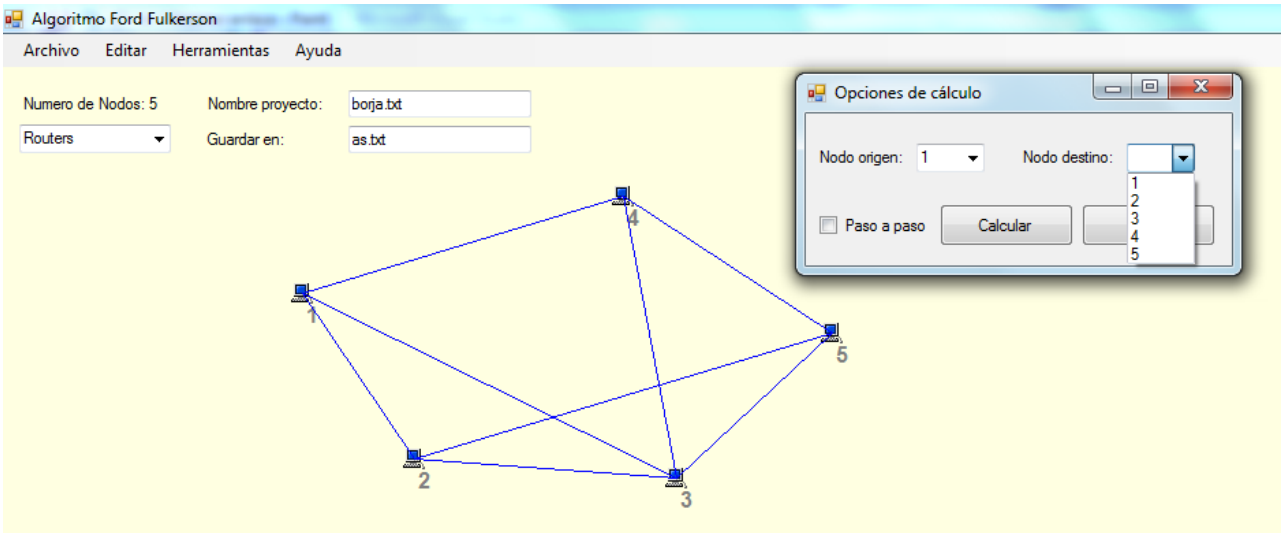


Figura 5.8. Opciones de cálculo

En la figura 5.8. vemos el menú opciones de cálculo que aparece al ejecutar la opción calcular. Este nos deja elegir el nodo origen y el nodo destino del cálculo que queremos realizar, y del mismo modo, nos deja marcar la opción 'paso a paso'. Esta opción ejecutará el algoritmo mostrando en pantalla cada camino que encuentra el algoritmo y finalmente nos dará la solución final. Si no marcamos esta opción, el algoritmo se ejecutará directamente y nos dará la solución final.

5.4. Salida de resultados

Una vez se han elegido las opciones de cálculo, se ejecuta el algoritmo. Así, este hallará todas las rutas de penetración existentes en la red con su correspondiente flujo y calculará el flujo máximo a través de la red. A continuación vemos el archivo que genera el algoritmo una vez finalizado:

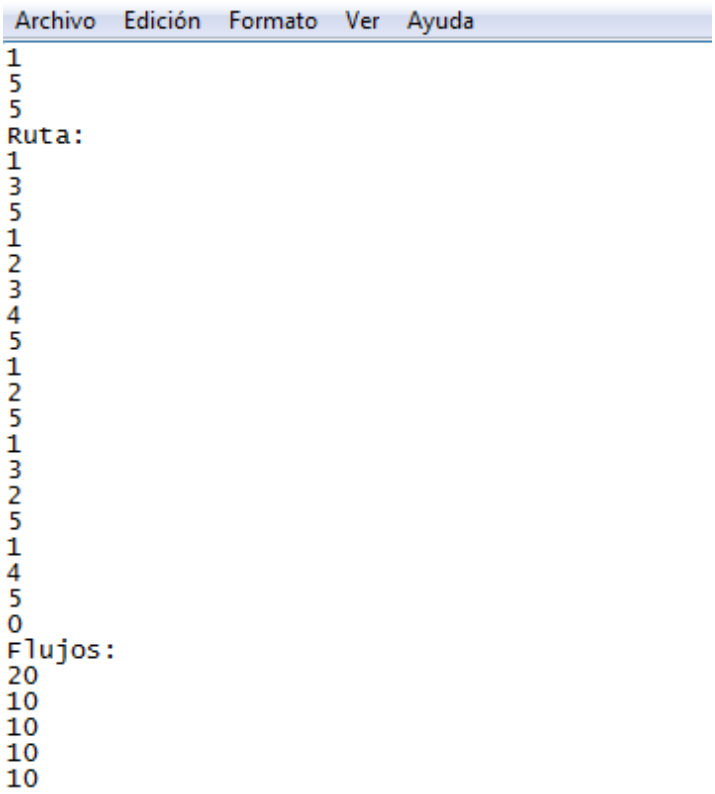


Figura 5.9. Archivo solución

En este archivo, se almacena en primer lugar el nodo origen y el nodo destino, en este caso nodos 1 y 5. En la línea siguiente se almacena el número de caminos encontrados. Después de la etiqueta 'rutas', encontramos los caminos encontrados, y después de la etiqueta 'flujos' encontramos el flujo de cada camino en orden de aparición.

A continuación veremos gráficamente el resultado que se obtiene cuando se elige la opción paso a paso en las opciones de cálculo.

En la figura 5.10. se ve como el algoritmo muestra el primer camino encontrado. En la ventana que aparece, se indica el número de camino, la ruta que sigue, y el flujo que circula por él. Vemos también la pestaña Siguiente paso, que continua ejecutando el algoritmo y muestra el siguiente camino.

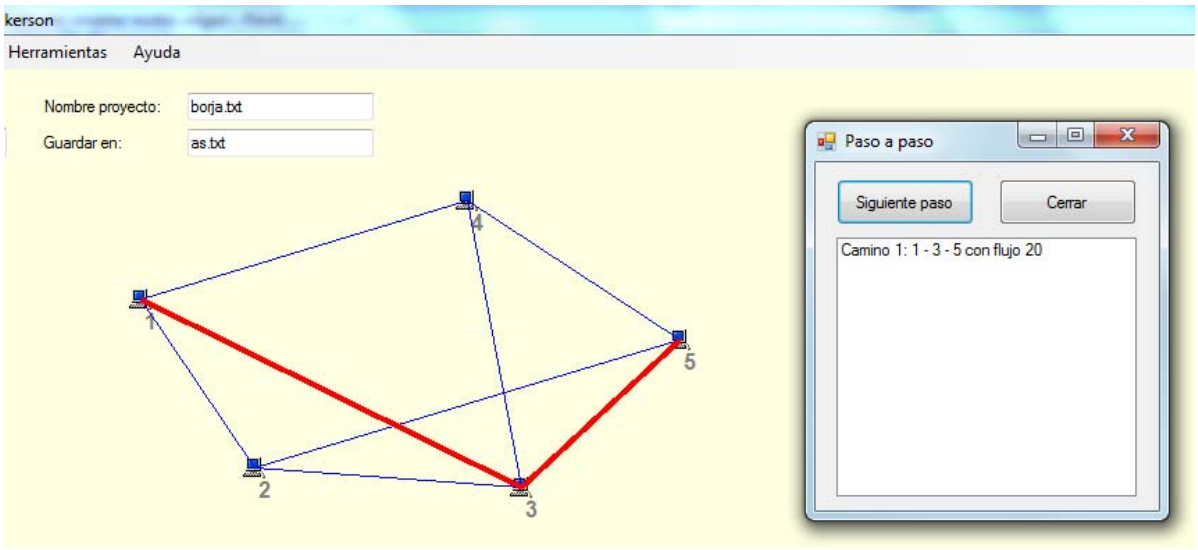


Figura 5.10. Primer camino encontrado

En cambio, si no elegimos la opción 'paso a paso', el algoritmo se ejecutará y nos mostrará una ventana con un resumen del proceso. E la figura 5.11. vemos ese resumen.

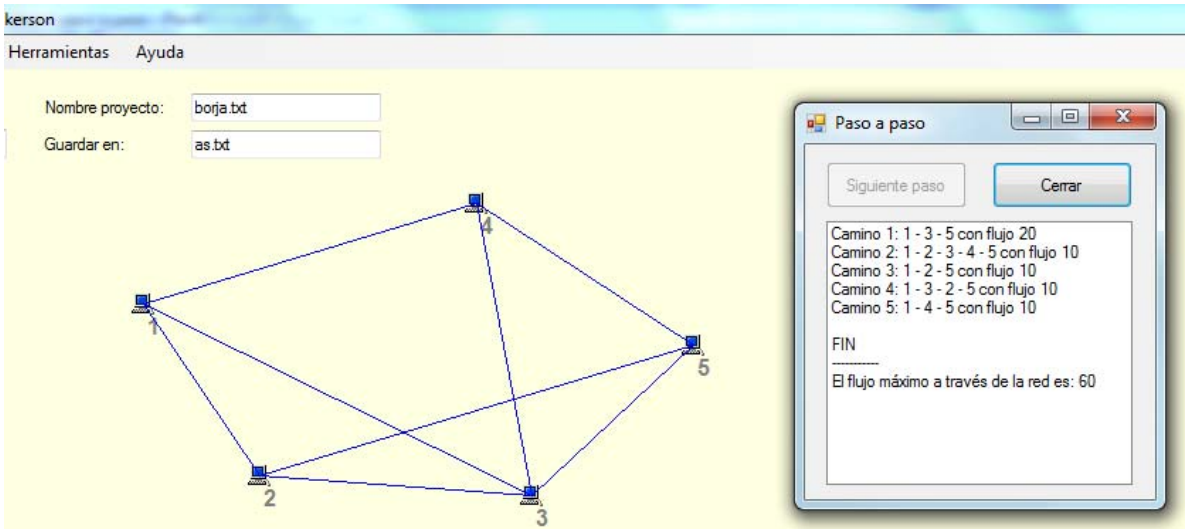
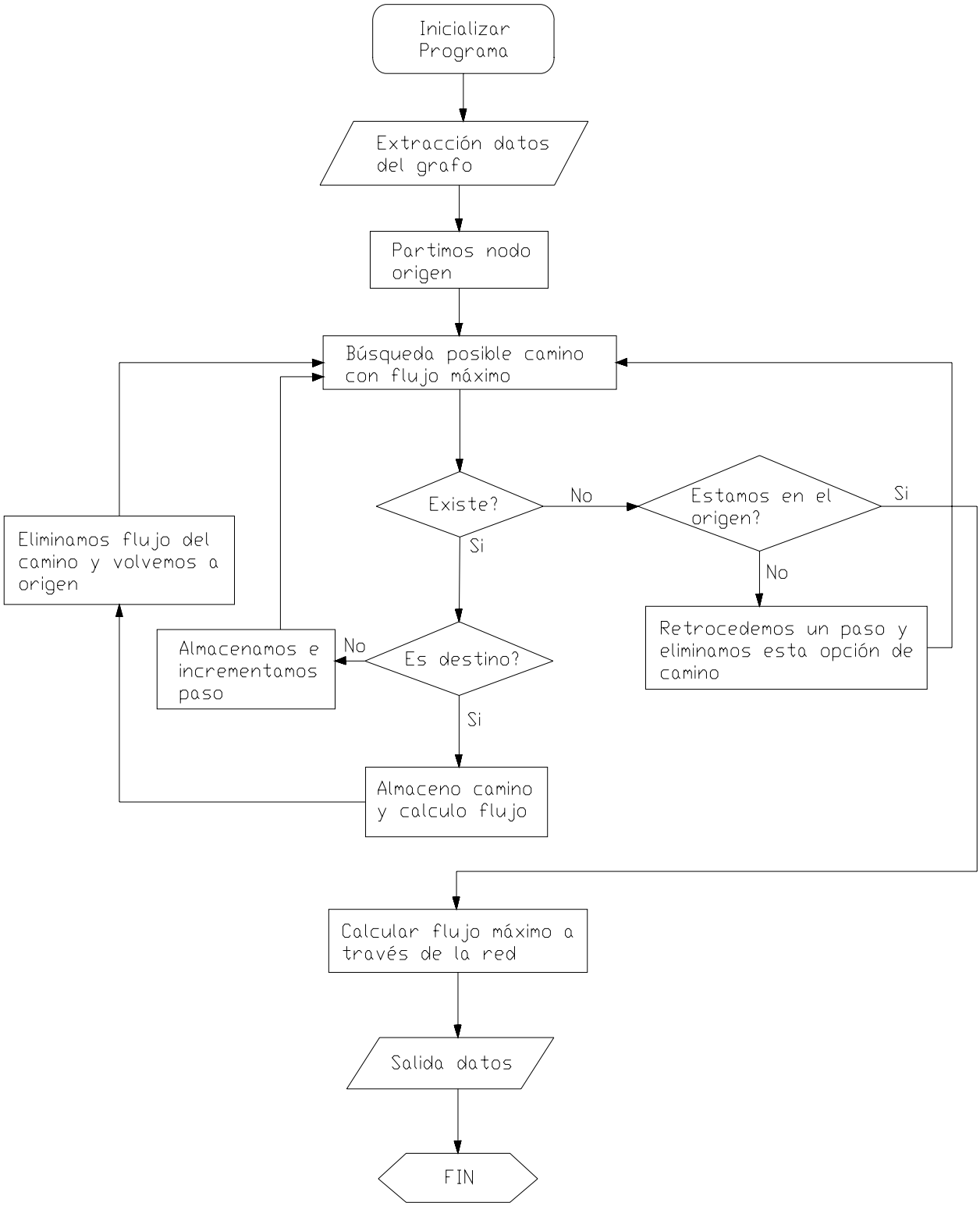


Figura 5.11. Resumen cálculo

En el resumen aparecen todos los caminos encontrados, con el flujo correspondiente, y para acabar el flujo máximo a través de la red.

5.5. Diagrama de flujo

El diagrama de flujo de este programa es el siguiente:



6. Ejemplos prácticos

Con el programa creado podemos analizar cualquier red que previamente definamos. Después de estudiar el algoritmo de Ford y Fulkerson queda claro que no puede ser aplicado a cualquier red.

Vamos a ver dos ejemplos al detalle de redes definidas en el programa y de la aplicación del algoritmo paso a paso.

6.1. Ejemplo 1: Red de 10 ordenadores de una pequeña empresa

El primer ejemplo consistirá en una red de intercambio de datos entre ordenadores en una pequeña empresa.

Analizaremos el funcionamiento en una red con pocos ordenadores, para ver el funcionamiento del algoritmo en una red sencilla, en la que las soluciones conociendo los flujos en cada arista se pueden extraer sin mucha dificultad.

Se considerará que en la empresa hay 10 ordenadores, y que estos pueden intercambiar datos entre sí.

Queremos obtener el flujo máximo de transmisión de datos del ordenador 1 al ordenador 10. El programa encuentra 5 caminos, y el flujo total a través de la red toma valor 60, en nuestro caso kbytes / seg.

En el Anejo 2 vemos el archivo que almacena esta red, y en el Anexo 4 vemos la solución del algoritmo paso a paso. Primero se mostrará el archivo solución que crea el programa al ser ejecutado el algoritmo, y después veremos paso a paso los caminos o rutas de penetración que va encontrando el algoritmo.

6.2. Ejemplo 2: Red de 25 satélites en órbita

El segundo ejemplo será más complejo y consistirá en una red de satélites en órbita alrededor de la Tierra que transmiten datos entre ellos.

Ahora se analizará el funcionamiento del algoritmo en una red con más elementos. Aquí las soluciones no se obtienen de forma tan trivial, y podemos obtener respuestas más sorprendentes.

Se considerará que la red está formada por 25 satélites, y que al estar en órbita alrededor de la Tierra, no todos pueden intercambiar datos entre sí.

Queremos obtener el flujo máximo de transmisión de datos del satélite 1 al satélite 12. El programa encuentra 6 caminos, y el flujo total a través de la red es de 55kb/seg.

En el Anejo 3 vemos el archivo que almacena esta red, y en el Anexo 5 vemos la solución del algoritmo paso a paso. Primero se mostrará el archivo solución que crea el programa al ser ejecutado el algoritmo, y después veremos paso a paso los caminos o rutas de penetración que va encontrando el algoritmo.

7. Conclusiones

Tras el estudio realizado hemos comprobado que toda la teoría de grafos es una herramienta que nos permite un amplio y preciso estudio de cualquier propiedad de una red, siendo aplicable a todos los ámbitos de la Ingeniería Civil.

Por tanto, queda patente la utilidad de la teoría de grafos, y concretamente dentro de ésta, de los algoritmos que resuelven el problema del camino mínimo.

Existen muchos algoritmos para hallar rutas en una red. En esta tesina se analizan algunos de ellos y se profundiza en el estudio del algoritmo de Ford y Fulkerson, especialmente sugestivo porque calcula las rutas de flujo máximo. El análisis del mencionado algoritmo es el núcleo de este trabajo.

Dada la inexistencia de algún programa que ejecute el algoritmo de Ford y Fulkerson, he querido crear una aplicación para estudiarlo amplia y extensamente, y posteriormente, para poderlo aplicar a redes de diferentes tamaños.

El desarrollo de esta aplicación, “Algoritmo de Ford-Fulkerson”, me ha creado la necesidad de analizar y profundizar sobre la teoría de grafos. Los problemas que se fueron manifestando progresivamente a lo largo del desarrollo mediante el sistema de “prueba y error”, se analizaron mediante la ejecución del programa paso a paso para detectar el error en cada caso.

Esto mismo provocó que para cada error que se producía tuviera que entender cómo había reaccionado el algoritmo, cómo se esperaba que respondiera y qué opciones estaban permitidas en la red sobre la que trabajaba, entendiendo ésta como un grafo. Así, a la vez que creaba el programa he ido analizando en profundidad el funcionamiento del algoritmo.

Líneas futuras de investigación

Existen muchas líneas abiertas para futuras investigaciones simplemente modificando o dando alguna limitación al algoritmo de Ford y Fulkerson. Por ejemplo, sería interesante modificarlo para que calculara caminos de máximo flujo que pasaran obligatoriamente por un punto determinado. También se podría limitar el número de aristas que queremos que recorra para llegar del nodo al destino, o limitar el flujo que queremos transportar.

Podríamos plantearnos el combinar el problema del flujo máximo con las ventanas de tiempo, y aunque en ese caso las utilidades del algoritmo se verían minimizadas, en algunos casos sería muy interesante aplicar este concepto de espera en un nodo para calcular añadir un nuevo criterio a la hora de elegir una ruta u otra.

En resumen, que podemos modificar de muchas maneras el algoritmo de Ford y Fulkerson y obtener resultados interesantes, se trata de modificaciones para cada ejemplo concreto, y que con el programa que he diseñado se podrían aplicar fácilmente.

8. Notas bibliográficas

La bibliografía dedicada a la Teoría de grafos es muy extensa, y abarcar todos los textos es imposible. Para la realización de esta tesina me he basado principalmente en aquellos libros y artículos que mi tutora me ha recomendado así como en los que se recomendaron en las clases durante la carrera en la Escuela. He consultado en todo momento sobretodo como punto de partida los apuntes disponibles de la asignatura de Transports sobre la Teoría de Grafos.

Como motivación de la tesina y punto de partida para conocer esta teoría, se ha utilizado el texto [11].

Para la redacción de los apartados destinados a la definición de los elementos básicos que forman la Teoría de grafos así como los distintos tipos de grafos se han utilizado los textos [1], [2], [3] i [4]. Las referencias a los distintos algoritmos de búsqueda estudiados se han usado los textos [6], [5], [9], [10] y [12].

Además se han consultado distintas páginas web que se citan como nota a pie de página en los apartados correspondientes. Estas son una página donde se puede editar un grafo y ver paso a paso como corre el algoritmo de Dijkstra, y otra donde encontramos el ejemplo del algoritmo de Ford y Fulkerson del punto 5.

9. Bibliografía

[1] Basart i Muñoz, J.M. (1994) *Grafs: fonaments i algoritmes*. 223 páginas. Universitat Autònoma de Barcelona. Manuals de la Universitat Autònoma de Barcelona, 13. ISBN: 84-7929-982-7.

[2] Comellas, F., Fàbrega, J., Sànchez, A., Serra, O. (2001). *Matemática discreta*. 336 páginas. Edicions UPC. ISBN: 84-8301-456-4.

[3] Chartrand, G., Lesniak, L. (1996). *Graphs and digraphs*. Ed Chapman & Hall

[4] Wilson, R.J. (1983). *Introduction to Graph Theory*. Ed Longman.

[5] Evans, J., Minieka, E. (1992). Optimization Alhorithms for Networks and Graphs..470 páginas. Marcel Dekker, Inc. ISBN: 0-8247-8602-5.

[6] Minoux, M., Bartnik, G. (1986) *Graphes, algoritmes logiciels*. Ed. Bordes.

[7] Gibbons, A. (1985). *Algorithmic Graph Theory*. Cambridge University Press.

[8] Gondran, M., Minoux, M. (1979). *Graphes et algoritmes*. Ed Eyrolles.

[9] Chen, Y.-L-, Yang, H.-H. (2000). Shortest paths in traffic-light networks. Transportation Research, nº34, 241-253.

[10] Pallotino, S., Scutella, M.G. (1998). Shortest paths algorithms in transportation models: Classical and innovative aspects. Equilibrium and advanced transportation modeling, 245-281.

[11] Balbuena, C. (2008) 'Grafos y redes de transporte'. Apuntes de la asignatura Transports II. Escola de Camins, Canals i Ports de Barcelona, UPC.

[12] Brunat Blay, J.M. (1996). Combinatòria i teoría de grafs. Aula teórica 46. Ediciones UPC. 104 páginas. ISBN: 84-8301-216-2.

ANEXOS

ANEXO 1. CÓDIGO DEL PROGRAMA

1. FORM 1

```
'Antes de empezar llamamos a las bibliotecas que utilizaremos

Imports System.Drawing
Imports System.Drawing.Drawing2D
Imports System.Windows.Forms
Imports System.IO

'Definimos las variables globales que utilizaremos.

Public Class Form1
    Public Structure Tipo_Nodo
        Public PosicionX As Integer
        Public PosicionY As Integer
    End Structure


    Public Nodo(1000) As Tipo_Nodo
    Public NumeroNodos As Integer
    Public Ramas(1000, 1000) As Integer
    Public NumeroRamas As Integer
    Public PasoPaso As Integer

    Public Estado As Integer

    Dim FirstPoint As Point
    Dim icon1 As New System.Drawing.Icon("router.ico")
    Dim icon2 As New System.Drawing.Icon("sat.ico")
    Dim Relleno As SolidBrush
    Dim Fuente As Font

    Public Routers(100) As System.Drawing.Graphics
    Public NumRouter(100) As System.Drawing.Graphics
    Public Lineas(500) As System.Drawing.Graphics
    Public LineasAux(500) As System.Drawing.Graphics

    'Función que cambia el cursor en modo insertar nodo

    Public Sub AnadirRama(ByVal a As Integer)
        Estado = 1
        Me.Cursor = Cursors.Cross

    End Sub
    'Inicialización de variables

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        NumeroNodos = 0
        NumeroRamas = 0
```

```
        Me.TextGuardar.Text = ""
        Me.TextSolucion.Text = ""


    End Sub
    'Actualiza las coordenadas del puntero

    Private Sub Form1_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseMove
        FirstPoint = e.Location
        ToolStripStatusLabel1.Text = FirstPoint.X
        ToolStripStatusLabel2.Text = FirstPoint.Y
    End Sub

    Private Sub RuterToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
RuterToolStripMenuItem.Click
        Estado = 1
        Me.Cursor = Cursors.Cross

    End Sub
    'Función de dibujo

    Public Sub Dibujar()
        Dim i As Integer
        Dim j As Integer
        Dim k As Integer
        Dim aux As String
        Dim Lapiz As Pen


        Me.Refresh()

        Lapiz = New Pen(Color.Blue)
        Relleno = New SolidBrush(Color.Gray)
        Fuente = New Font("Arial", 12, FontStyle.Bold)
        For i = 1 To NumeroNodos Step 1

            Routers(i) = Me.CreateGraphics
            If ComboBox1.SelectedItem = "Routers" Then
                Routers(i).DrawIcon(icon1, Nodo(i).PosicionX,
Nodo(i).PosicionY)
            End If
            If ComboBox1.SelectedItem = "Satélites" Then
                Routers(i).DrawIcon(icon2, Nodo(i).PosicionX,
Nodo(i).PosicionY)
            End If

            NumRouter(i) = Me.CreateGraphics
            NumRouter(i).DrawString(i, Fuente, Relleno,
Nodo(i).PosicionX + 8, Nodo(i).PosicionY + 15)

        Next
```

```
        For i = 0 To NumeroNodos
            For j = 0 To NumeroNodos
                If Ramas(i, j) <> 0 Then
                    Lineas(k) = CreateGraphics()
                    Lineas(k).DrawLine(Lapiz, Nodo(i).PosicionX + 8,
Nodo(i).PosicionY + 8, Nodo(j).PosicionX + 8, Nodo(j).PosicionY + 8)
                    k = k + 1
                End If
            Next
        Next
    End Sub
    'Inserción de nodo

    Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseDown


        FirstPoint = e.Location
        If Estado = 1 Then
            NumeroNodos = NumeroNodos + 1
            Relleno = New SolidBrush(Color.Gray)
            Fuente = New Font("Arial", 12, FontStyle.Bold)
            Nodo(NumeroNodos).PosicionX = FirstPoint.X
            Nodo(NumeroNodos).PosicionY = FirstPoint.Y
            Text1.Text = "Numero de Nodos: " & NumeroNodos

            Call Dibujar()

            Estado = 0
            Me.Cursor = Cursors.Default
        End If



    End Sub
    'Botón insertar enlace, llama a form2

    Private Sub EnlaceToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
EnlaceToolStripMenuItem.Click

        Form2.Show()



    End Sub
    'Botón salir

    Private Sub SalirToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
SalirToolStripMenuItem.Click
        Me.Close()
    End Sub
```

```
    Private Sub Label1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)

    End Sub
    'Función guardar el grafo definido

    Private Sub GuardarToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
GuardarToolStripMenuItem.Click
        Dim file As System.IO.StreamWriter
        Dim i As Integer
        Dim j As Integer

        file =
My.Computer.FileSystem.OpenTextFileWriter(TextGuardar.Text, True)
        file.WriteLine(NumeroNodos)
        For i = 1 To NumeroNodos Step 1
            file.WriteLine(Nodo(i).PosicionX)
            file.WriteLine(Nodo(i).PosicionY)
        Next
        file.WriteLine(NumeroRamas)
        For i = 1 To NumeroNodos
            For j = 1 To NumeroNodos
                If Ramas(i, j) <> 0 Then
                    file.WriteLine(i)
                    file.WriteLine(j)
                    file.WriteLine(Ramas(i, j))

                End If
            Next
        Next
        file.WriteLine("-1")
        file.Close()


    End Sub
    'Función abrir archivo

    Private Sub FdgdToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
FdgdToolStripMenuItem.Click
        Dim file As System.IO.StreamReader
        Dim i As Integer
        Dim j As Integer
        Dim x As Integer
        Dim y As Integer
        Dim openFileDialog1 As New OpenFileDialog()

        For i = 1 To NumeroNodos
            For j = 1 To NumeroNodos
                Ramas(i, j) = 0
            Next
```

```
        Next
        file =
My.Computer.FileSystem.OpenTextFileReader(TextGuardar.Text)
        NumeroNodos = file.ReadLine()
        For i = 1 To (NumeroNodos)
            Nodo(i).PosicionX = file.ReadLine()
            Nodo(i).PosicionY = file.ReadLine()
        Next
        NumeroRamas = CInt(file.ReadLine())
        x = file.ReadLine()
        While x <> -1
            y = file.ReadLine()
            Ramas(x, y) = file.ReadLine()
            x = file.ReadLine()
        End While

        Call Dibujar()


        Text1.Text = "Numero de Nodos: " & NumeroNodos

    End Sub
    'Función Nuevo archivo

    Private Sub NuevoToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
PeneToolStripMenuItem.Click
        Dim i As Integer
        Dim j As Integer

        NumeroNodos = 0
        NumeroRamas = 0
        Me.TextGuardar.Text = ""
        Me.TextSolucion.Text = ""

        For i = 1 To NumeroNodos
            For j = 1 To NumeroNodos
                Ramas(i, j) = 0
            Next
        Next
        Me.Refresh()
        Text1.Text = "Numero de Nodos: " & NumeroNodos

    End Sub
    'Botón calcular, llama a form 3

    Private Sub CalcularToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
CalcularToolStripMenuItem.Click
        Form3.Show()
    End Sub
```

```
    Private Sub TextGuardar_TextChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles TextGuardar.TextChanged
        Call Dibujar()

    End Sub

    Private Sub TextSolucion_TextChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
TextSolucion.TextChanged
        Call Dibujar()

    End Sub

    Private Sub Text1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Text1.Click

    End Sub

    Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
ComboBox1.SelectedIndexChanged
        Call Dibujar()

    End Sub
    'Definición botón Ayuda

    Private Sub
AcercaDeAlgoritmoFordFulkersonToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
AcercaDeAlgoritmoFordFulkersonToolStripMenuItem.Click
        MsgBox("Este programa ejecuta el algoritmo de Ford y Fulkerson
para cualquier grafo que definamos." & Chr(13) & "Se ha limitado el
número de nodos a 1000" & Chr(13) & Chr(13) & "Autor del programa:
Borja Muñoz Echevarría" & Chr(13) & "2010")

    End Sub
End Class
```

2. FORM 2

```
Imports System.Drawing
Imports System.Drawing.Drawing2D
Imports System.Windows.Forms
'Definimos las opciones de enlace en función de los nodos definidos

Public Class Form2

    Private Sub Form2_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        Form1.Dibujar()
        Dim i As Integer
        For i = 1 To Form1.NumeroNodos
            ComboOrigen.Items.Add(CStr(i))
            ComboDestino.Items.Add(CStr(i))
        Next

    End Sub
    'Botón aceptar

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button1.Click
        Me.Close()
        Form1.Dibujar()
    End Sub

    Private Sub TextBox1_TextChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs)

    End Sub
    'Función definir enlace

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button2.Click
        Dim Lapiz As Pen
        Form1.NumeroRamas = Form1.NumeroRamas + 1
        Lapiz = New Pen(Color.Blue)
        Form1.Lineas(1) = Form1.CreateGraphics
        Form1.Lineas(1).DrawLine(Lapiz,
Form1.Nodo(CInt(ComboOrigen.Text)).PosicionX + 8,
Form1.Nodo(CInt(ComboOrigen.Text)).PosicionY + 8,
Form1.Nodo(CInt(ComboDestino.Text)).PosicionX + 8,
Form1.Nodo(CInt(ComboDestino.Text)).PosicionY + 8)
        Form1.Ramas(CInt(ComboOrigen.Text), CInt(ComboDestino.Text)) =
CInt(TextCap1.Text)
        Form1.Ramas(CInt(ComboDestino.Text), CInt(ComboOrigen.Text)) =
CInt(TextCap2.Text)

        Me.Close()
        Call Form1.Dibujar()

    End Sub
```

```
    Private Sub Form2_Move(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Move
        Form1.Dibujar()
    End Sub

    Private Sub Label1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Label1.Click

    End Sub

    Private Sub ComboOrigen_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
ComboOrigen.SelectedIndexChanged

    End Sub



    Private Sub ComboDestino_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
ComboDestino.SelectedIndexChanged

    End Sub
End Class
```

3. FORM 3

```
Public Class Form3

    'Llamamos al algoritmo en función de si se ha marcado paso a paso.

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button1.Click

        Call Calcular()
        Form1.PasoPaso = 0
        If CheckBox1.Checked Then
            Form1.PasoPaso = 1
        End If

        Form4.Show()

        Me.Close()

    End Sub

    'Algoritmo de Ford y Fulkerson

    Public Sub Calcular()
        Dim RamasTemp(100, 100) As Integer
        Dim CaminosTemp(1000) As Integer
        Dim FlujosTemp(1000) As Integer
        Dim PunteroCaminos As Integer
        Dim PunteroFlujos As Integer


        Dim file3 As System.IO.StreamWriter
        file3 =
My.Computer.FileSystem.OpenTextFileWriter(Form1.TextSolucion.Text,
False)



        Dim i As Integer
        Dim j As Integer
        Dim k As Integer
        Dim MaxCap As Integer
        Dim MinCap As Integer
        Dim flag As Integer


        Dim Aux As Integer

        Dim Direccion(500) As Integer
        Dim DireccionError(100) As Integer
        Dim Flujo(500) As Integer
        Dim Estado As Integer
        Dim Errores As Integer
```

```
        Dim Origen As Integer
        Dim Destino As Integer
        Origen = ComboBox1.SelectedItem
        Destino = ComboBox2.SelectedItem

        If (Origen Or Destino) = 0 Then
            MsgBox("Debes seleccionar un nodo origen / destino")

        End If

        For i = 0 To 100
            For k = 0 To 100
                RamasTemp(i, k) = Form1.Ramas(i, k)
            Next
        Next

        PunteroCaminos = 0
        Direccion(1) = Origen
        Flujo(1) = 65000 ' Defino un maximo
        i = 1
        Errores = 0

        '-------------------------------------------------------------
-------
        While (Estado <> 1)

            MaxCap = 0
            Aux = 0
            For j = 1 To CInt(Form1.NumeroNodos)
                If RamasTemp(Direccion(i), j) <> 0 Then
                    flag = 1
                    For k = 1 To i
                        If Direccion(k) = j Then
                            flag = 0
                        End If
                    Next
                    For k = 1 To Errores
                        If DireccionError(k) = j Then
                            flag = 0
                        End If
                    Next

                    If ((MaxCap < RamasTemp(Direccion(i), j)) And
(flag = 1)) Then
                        MaxCap = RamasTemp(Direccion(i), j)
                        Aux = j
                    End If
                End If
            Next

            If MaxCap = 0 Then
```

```
                If i = 1 Then
                    Estado = 1
                Else
                    Errores = Errores + 1
                    DireccionError(Errores) = Direccion(i)
                    i = i - 1
                End If
            Else
                i = i + 1
                Direccion(i) = Aux
                Flujo(i) = MaxCap
                Errores = 0

            End If

            If Direccion(i) = Destino Then
                For k = 1 To i   ' Guardamos
                    CaminosTemp(PunteroCaminos) = Direccion(k)
                    PunteroCaminos = PunteroCaminos + 1
                Next
                MinCap = 65001 'flujo maximo
                For k = 1 To i 'Buscamos minimo
                    If Flujo(k) < MinCap Then
                        MinCap = Flujo(k)
                    End If
                Next
                FlujosTemp(PunteroFlujos) = MinCap
                PunteroFlujos = PunteroFlujos + 1


                For k = 1 To i - 1 ' Actualizar flujos
                    RamasTemp(Direccion(k), Direccion(k + 1)) =
RamasTemp(Direccion(k), Direccion(k + 1)) - MinCap
                Next
                For k = 1 To i - 1 ' Actualizar flujos
                    RamasTemp(Direccion(k + 1), Direccion(k)) =
RamasTemp(Direccion(k + 1), Direccion(k)) + MinCap
                Next
                Direccion(1) = Origen       'Inicializo variables
                Flujo(1) = 65000 'maximo
                i = 1

            End If


        End While
        file3.WriteLine(CStr(Origen))
        file3.WriteLine(CStr(Destino))
        file3.WriteLine(CStr(PunteroFlujos))
        file3.WriteLine("Ruta:")
        For k = 0 To PunteroCaminos
            file3.WriteLine(CStr(CaminosTemp(k)))
        Next
```

```
        file3.WriteLine("Flujos:")
        For k = 0 To PunteroFlujos
            file3.WriteLine(CStr(FlujosTemp(k)))
        Next
        file3.Close()


    End Sub
    'Inicializamos las opciones de nodo origen y destino en función
del número de nodos

    Private Sub Form3_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        Dim i As Integer
        For i = 1 To Form1.NumeroNodos
            ComboBox1.Items.Add(CStr(i))
            ComboBox2.Items.Add(CStr(i))
        Next
    End Sub

    Private Sub CheckBox1_CheckedChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
CheckBox1.CheckedChanged

    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button2.Click
        Me.Close()

    End Sub

    Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
ComboBox1.SelectedIndexChanged

    End Sub

    Private Sub ComboBox2_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
ComboBox2.SelectedIndexChanged

    End Sub
End Class
```

4. FORM 4

```
Public fin As Integer

    'Definimos las opciones calcular y calcular paso a paso

    Private Sub Form4_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        Call Form1.Dibujar()
        If Form1.PasoPaso = 1 Then
            paso = 1
            Call SiguientePaso()
        Else
            paso = 1
            While fin <> 1
                Call SiguientePaso()
                paso = paso + 1
            End While

        End If



    End Sub

    'Función siguiente paso.

    Private Sub SiguientePaso()
        Dim i As Integer
        Dim j As Integer
        Dim k As Integer
        Dim r As Integer


        Dim NumLineas As Integer
        Dim aux As String
        Dim Origen As Integer
        Dim Destino As Integer
        Dim NumCaminos As Integer
        Dim Lapiz As Pen
        Dim AuxString As String

        Lapiz = New Pen(Color.Red)
        Lapiz.Width = 4

        Call Form1.Dibujar()



        Dim file As System.IO.StreamReader
        file =
My.Computer.FileSystem.OpenTextFileReader(Form1.TextSolucion.Text)
        Origen = CInt(file.ReadLine())
```

```
        Destino = CInt(file.ReadLine())
        NumCaminos = CInt(file.ReadLine())
        aux = file.ReadLine()

        If paso <= NumCaminos Then

            Form1.LineasAux(5) = Form1.CreateGraphics()
            Form1.LineasAux(5).DrawLine(Pens.Red, 1, 1000, 1, 10)


            While i < paso
                r = CInt(file.ReadLine())
                If r = Origen Then
                    i = i + 1
                End If

            End While

            NumLineas = NumLineas + 1

            j = CInt(file.ReadLine())
            AuxString = "Camino " & CStr(paso) & ": " & CStr(Origen) &
" - " & CStr(j)

            Form1.LineasAux(NumLineas) = Form1.CreateGraphics()
            Form1.LineasAux(NumLineas).DrawLine(Lapiz,
Form1.Nodo(Origen).PosicionX + 8, Form1.Nodo(Origen).PosicionY + 8,
Form1.Nodo(j).PosicionX + 8, Form1.Nodo(j).PosicionY + 8)

            While j <> Destino
                k = CInt(file.ReadLine())
                AuxString = AuxString & " - " & CStr(k)
                NumLineas = NumLineas + 1
                Form1.LineasAux(NumLineas) = Form1.CreateGraphics()
                Form1.LineasAux(NumLineas).DrawLine(Lapiz,
Form1.Nodo(j).PosicionX + 8, Form1.Nodo(j).PosicionY + 8,
Form1.Nodo(k).PosicionX + 8, Form1.Nodo(k).PosicionY + 8)
                j = k

            End While

            While aux <> "Flujos:"
                aux = file.ReadLine()
            End While
            For j = 1 To paso
                aux = file.ReadLine()
            Next
            FluMax = FluMax + CInt(aux)
            AuxString = AuxString & " con flujo " & aux

            ListBox1.Items.Add(AuxString)
        Else
            Button1.Enabled = False
```

```
            ListBox1.Items.Add("   ")
            ListBox1.Items.Add("FIN")
            ListBox1.Items.Add("-----------")
            ListBox1.Items.Add("El flujo máximo a través de la red es:
" & CStr(FluMax))
            fin = 1
        End If


    End Sub

    'Botón siguiente paso

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button1.Click
        Form1.Refresh()
        paso = paso + 1

        Call Form1.Dibujar()


        Call SiguientePaso()

    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button2.Click
        Me.Close()

    End Sub

    Private Sub ListBox1_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
ListBox1.SelectedIndexChanged

    End Sub
End Class
```

ANEXO 2. EJEMPLO 1

Archivo de red:

10

211

181

293

302

485

316

446

110

599

210

690

334

667

108

839

217

887

361

984

171

14

1

2

20

1

91

3

30

1

4

10

2

3

40

2

5

30

3

4

10

3

5

20

3

6

5

3

8

20

4

3

5

4

5

20

4

92

7

10

5

6

20

5

7

30

5

8

30

6

3

20

6

5

10

6

7

20

6

8

30

6

9

10

7

4

10

7

93

5

10

7

6

20

7

8

40

7

10

40

8

3

10

8

5

20

8

7

40

8

9

30

8

10

40

9

6

2

9

94

8

30

9

10

50

10

7

10

10

8

40

10

9

10

-1

95

ANEXO 3. EJEMPLO 2

Archivo de red:

25

213

321

327

244

350

342

302

460

478

161

643

182

428

524

604

556

780

549

927

493

1016

424

1061

344

941

97

254

502

286

584

246

785

284

912

366

825

449

713

455

477

435

572

377

690

322

783

398

802

197

610

482

59

1

2

30

2

98

5

20

5

6

10

6

5

15

6

24

20

24

13

30

13

24

10

13

12

20

12

11

30

2

14

15

14

5

20

1

99

3

20

3

1

10

3

2

30

3

14

30

1

4

20

4

1

10

4

7

20

7

8

30

8

7

15

8

9

20

9

100

10

20

10

11

30

11

10

15

3

20

20

4

20

30

20

21

20

21

20

15

14

21

30

21

14

5

5

15

10

15

101

5

5

15

6

15

6

15

20

14

15

30

21

22

20

15

22

20

22

15

15

22

16

25

16

22

10

6

16

10

16

102

6

10

16

24

20

24

16

10

16

13

10

13

16

20

16

12

20

12

16

5

20

25

20

7

25

15

20

7

20

7

103

20

10

25

8

20

20

19

30

25

19

15

19

25

10

8

18

10

19

18

20

18

9

15

9

18

20

19

23

20

23

104

19

30

20

23

10

21

23

20

22

23

30

23

22

10

16

23

15

23

16

20

23

17

20

17

23

10

16

17

20

17

105

16

10

22

6

10

6

22

10

3

21

20

21

3

5

4

3

10

3

4

20

23

18

10

18

17

10

17

18

25

17

106

12

20

12

17

10

17

11

20

18

11

20

11

18

15

18

10

20

10

18

10

14

22

20

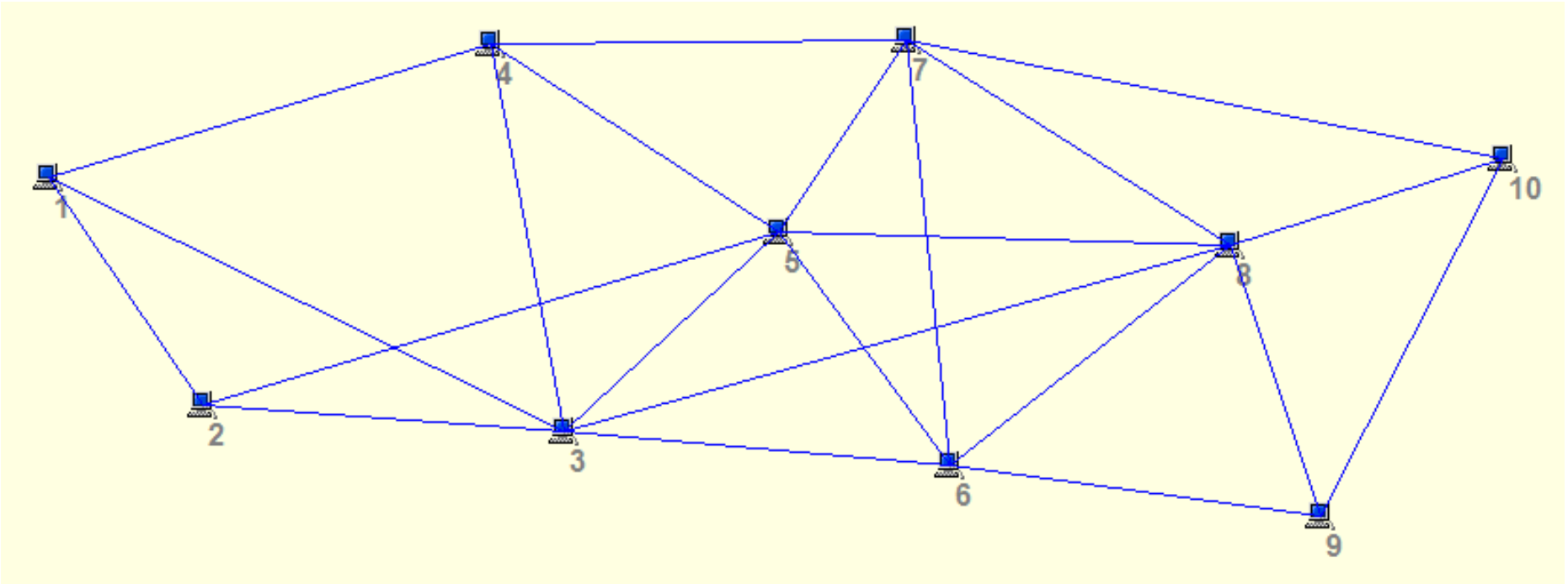
22

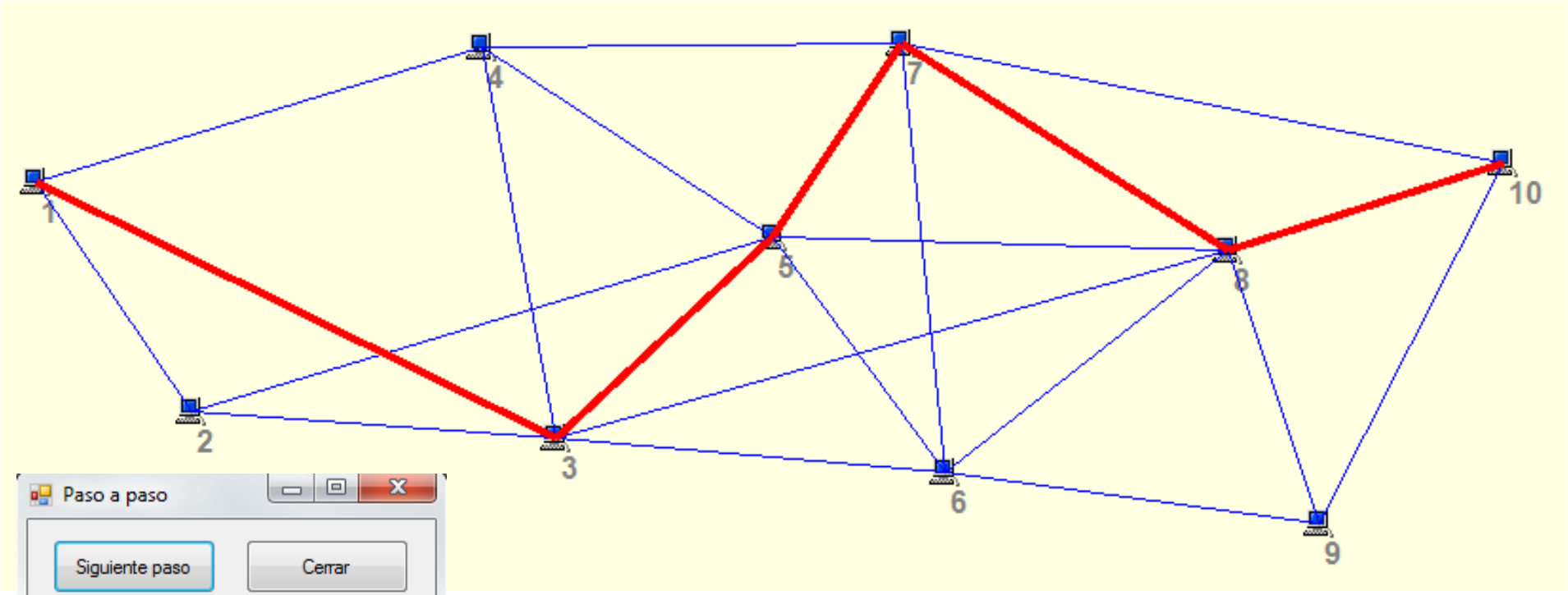
14

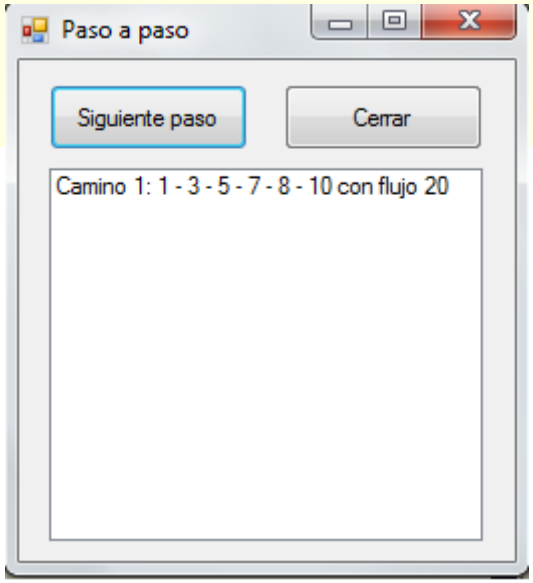
10

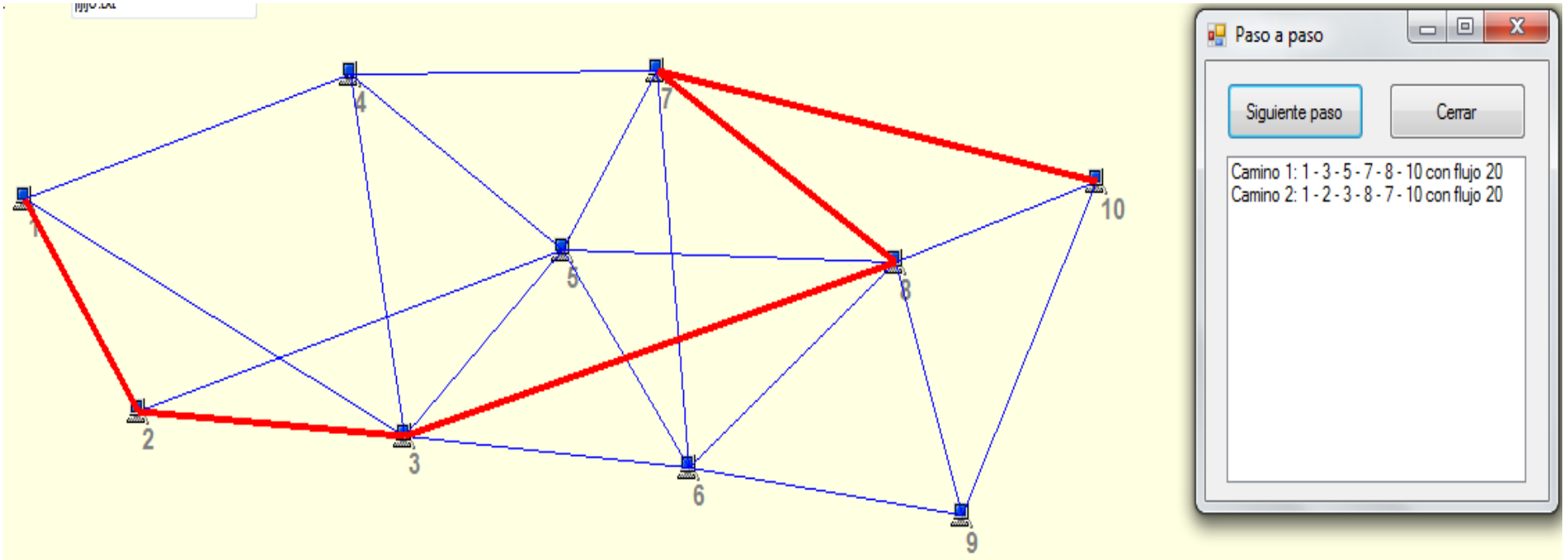
-1

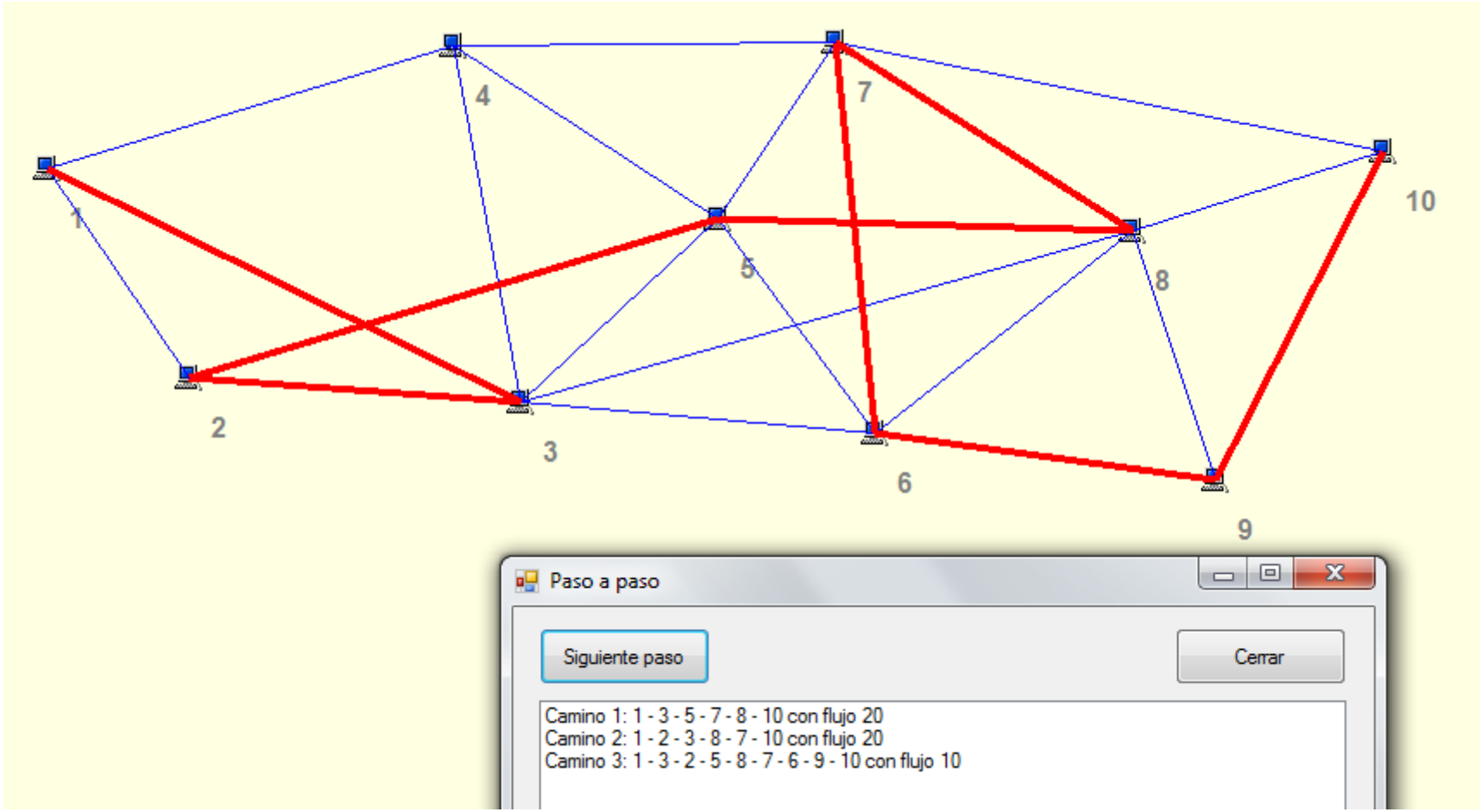
ANEXO 4. RESULTADOS EJEMPLO 1

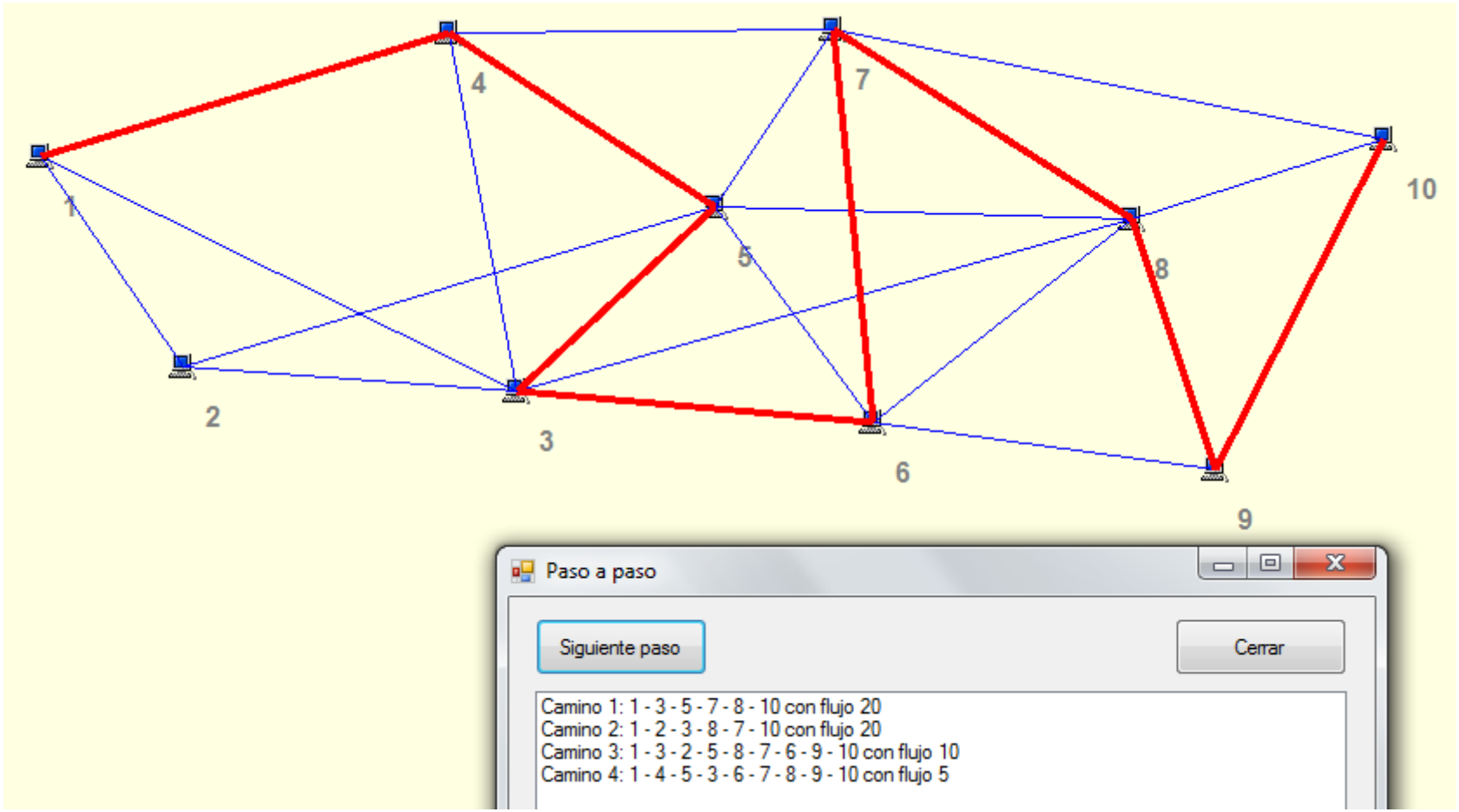


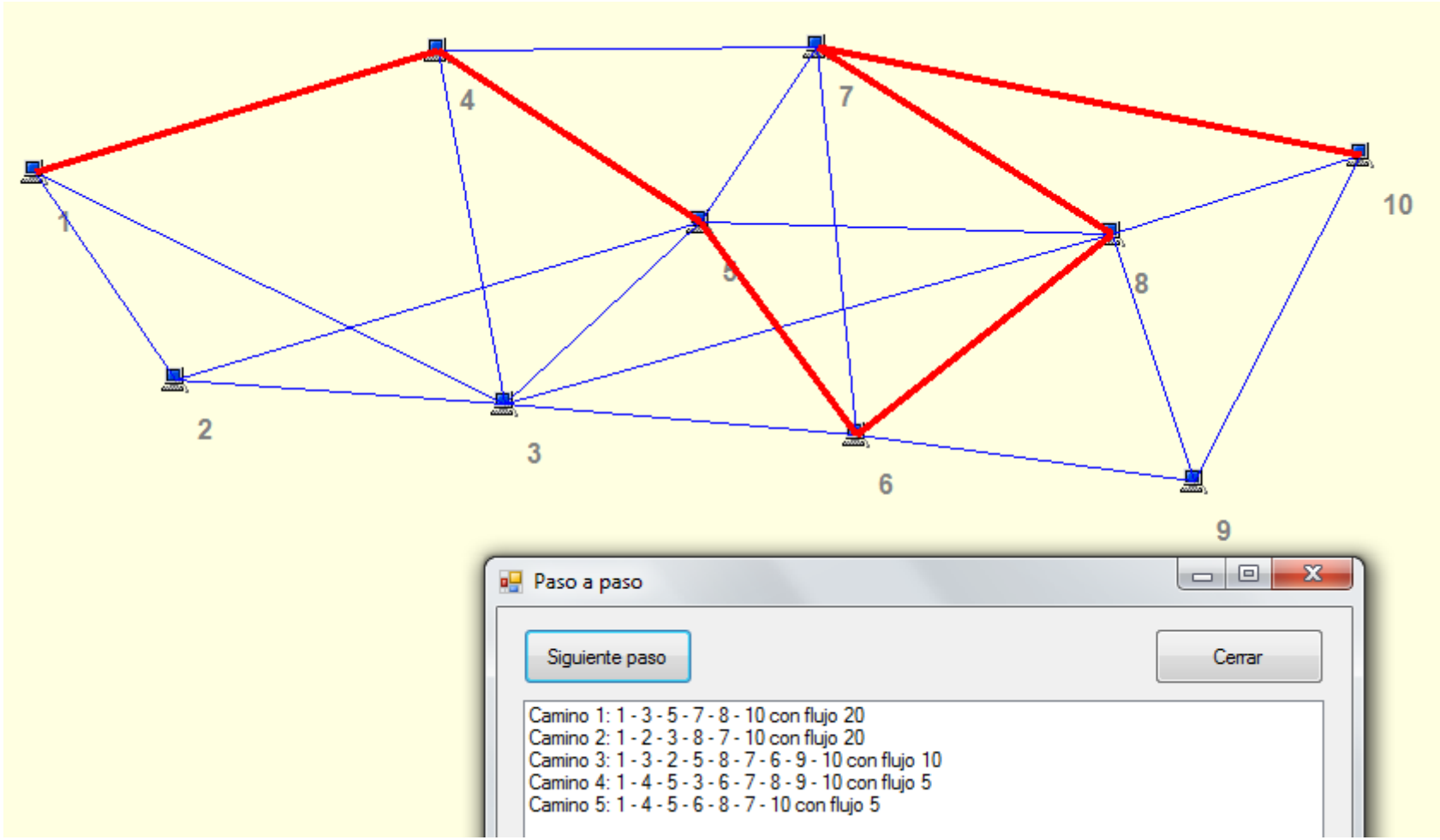

1
2
3
4
5
6
7
8
9
10

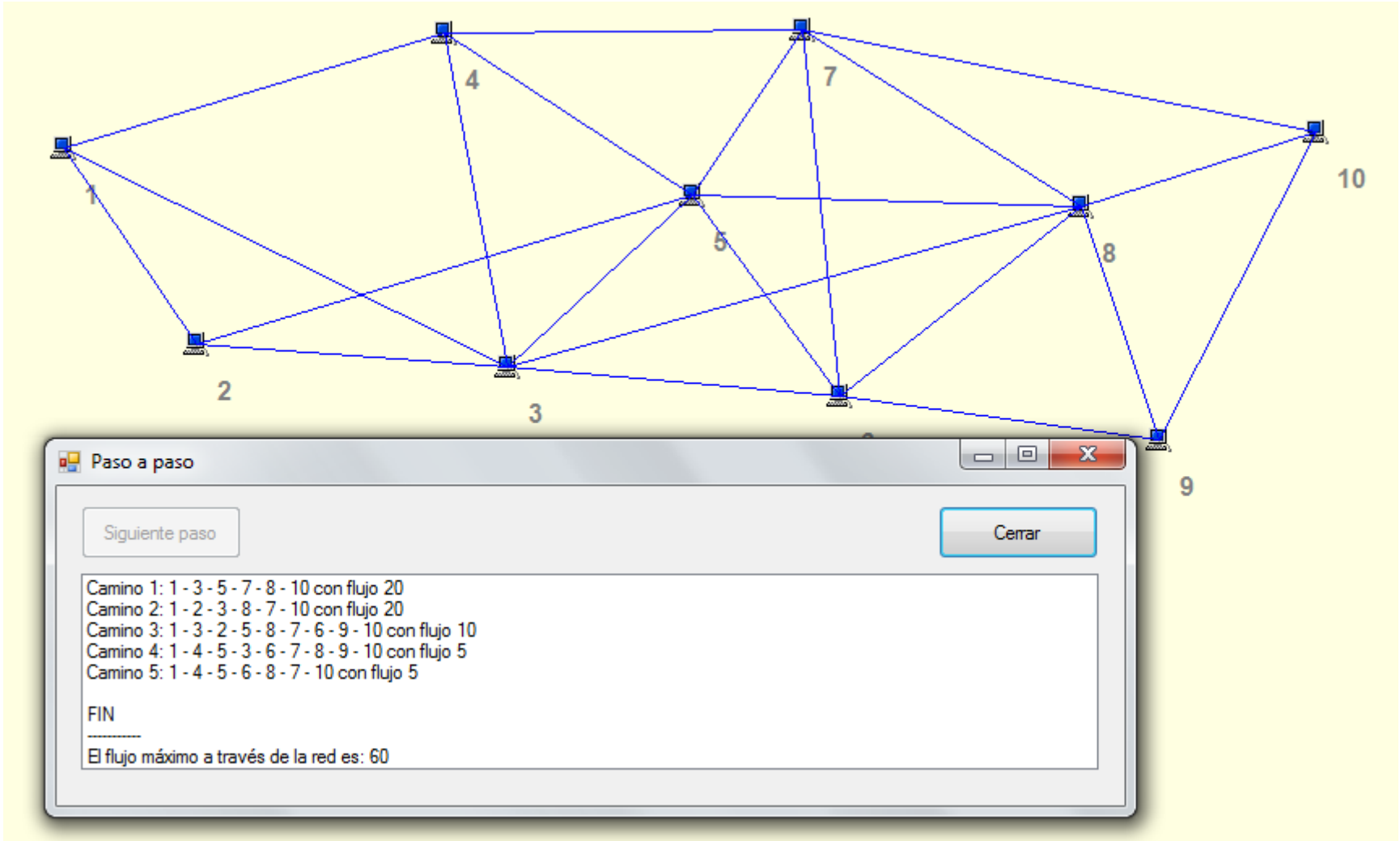

Paso a paso
Siguiente paso
Cerrar
Camino 1: 1 - 3 - 5 - 7 - 8 - 10 con flujo 20


1
2
3
4
5
6
7
8
9
10
Paso a paso
Siguiente paso
Cerrar
Camino 1: 1 - 3 - 5 - 7 - 8 - 10 con flujo 20
Camino 2: 1 - 2 - 3 - 8 - 7 - 10 con flujo 20


1
2
3
4
5
6
7
8
9
10
Paso a paso
Siguiente paso
Cerrar
Camino 1: 1 - 3 - 5 - 7 - 8 - 10 con flujo 20
Camino 2: 1 - 2 - 3 - 8 - 7 - 10 con flujo 20
Camino 3: 1 - 3 - 2 - 5 - 8 - 7 - 6 - 9 - 10 con flujo 10

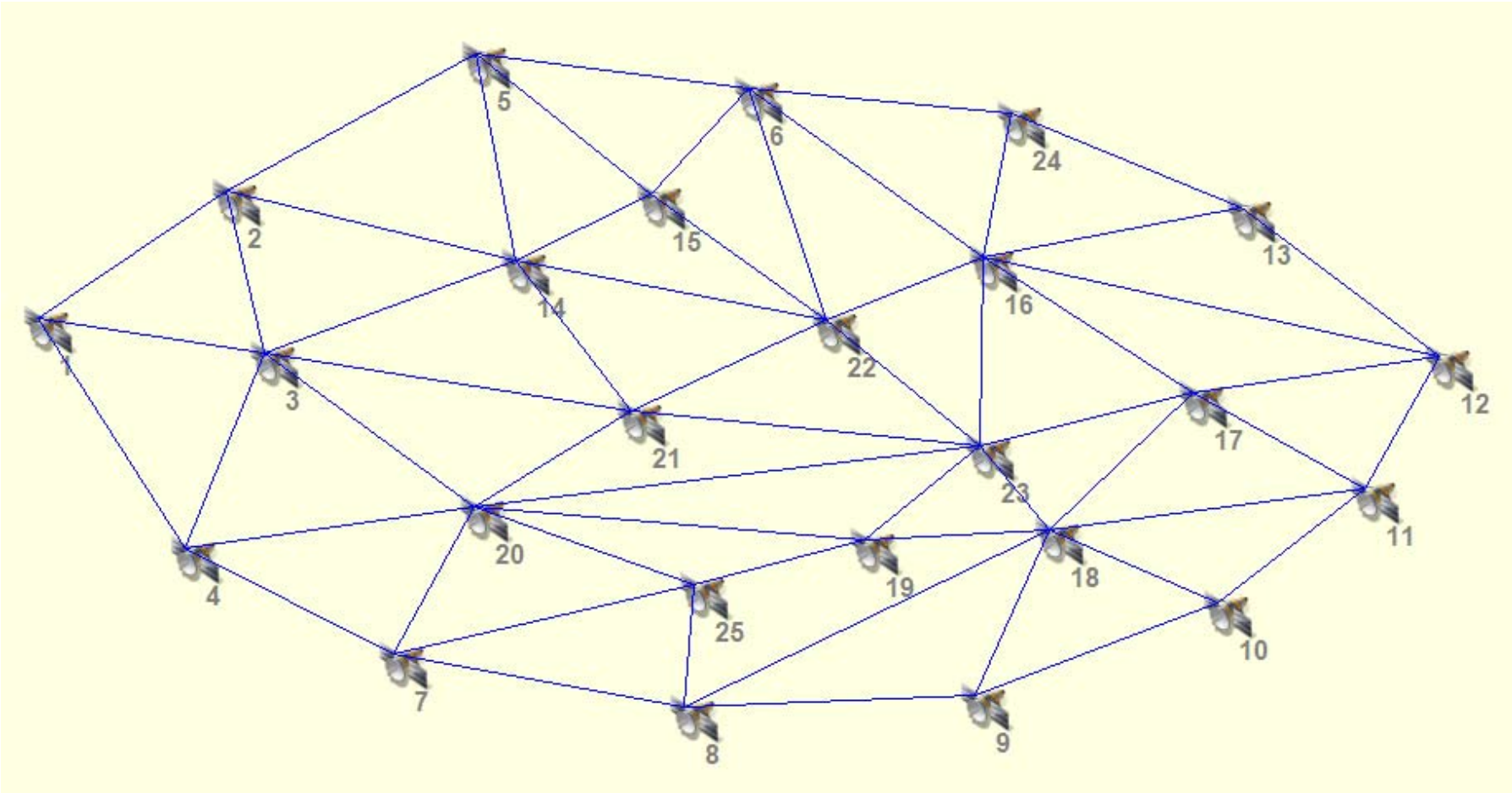

1
2
3
4
5
6
7
8
9
10
Paso a paso
Siguiente paso
Cerrar
Camino 1: 1 - 3 - 5 - 7 - 8 - 10 con flujo 20
Camino 2: 1 - 2 - 3 - 8 - 7 - 10 con flujo 20
Camino 3: 1 - 3 - 2 - 5 - 8 - 7 - 6 - 9 - 10 con flujo 10
Camino 4: 1 - 4 - 5 - 3 - 6 - 7 - 8 - 9 - 10 con flujo 5

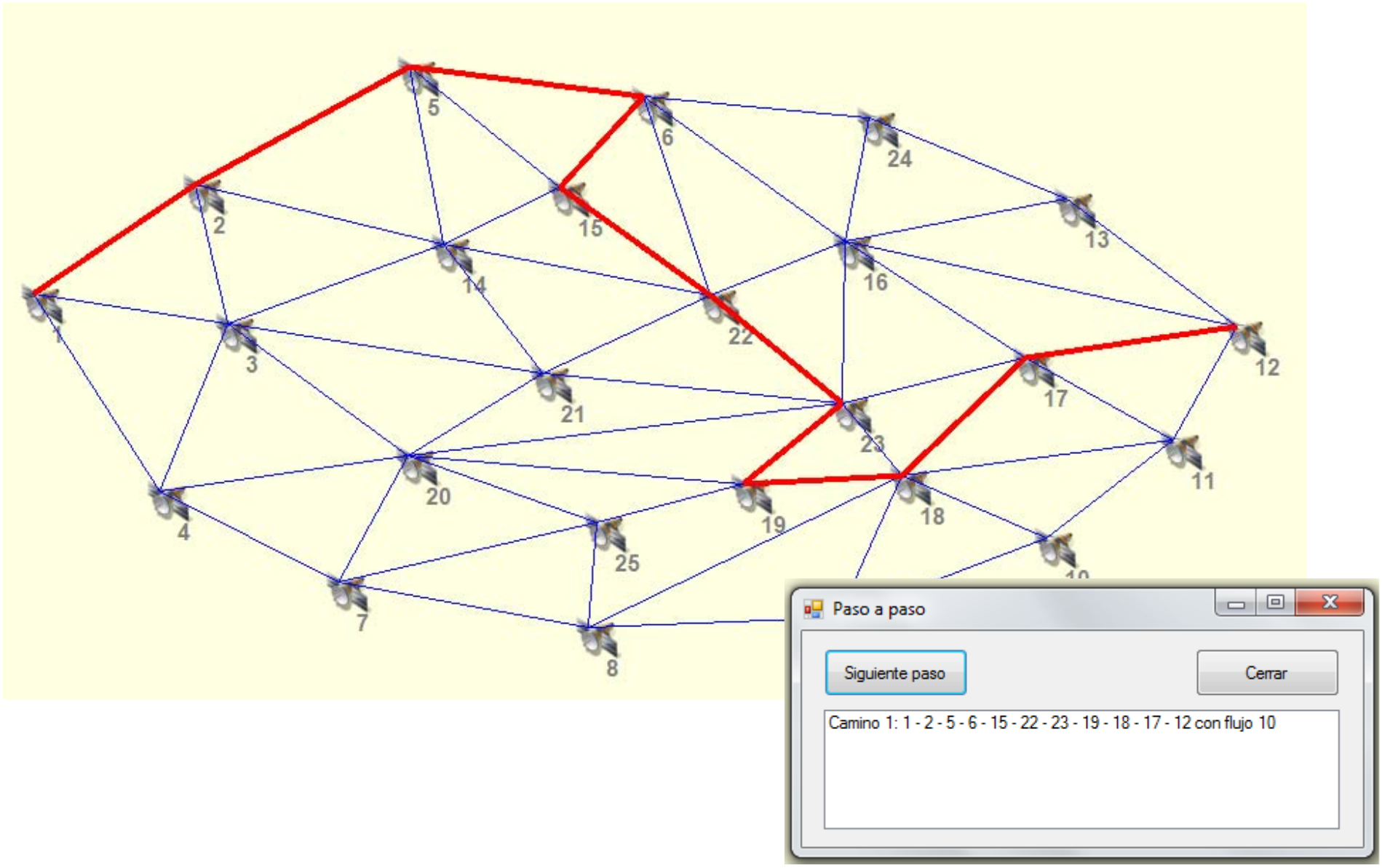

1
2
3
4
5
6
7
8
9
10
Paso a paso
Siguiente paso
Cerrar
Camino 1: 1 - 3 - 5 - 7 - 8 - 10 con flujo 20
Camino 2: 1 - 2 - 3 - 8 - 7 - 10 con flujo 20
Camino 3: 1 - 3 - 2 - 5 - 8 - 7 - 6 - 9 - 10 con flujo 10
Camino 4: 1 - 4 - 5 - 3 - 6 - 7 - 8 - 9 - 10 con flujo 5
Camino 5: 1 - 4 - 5 - 6 - 8 - 7 - 10 con flujo 5

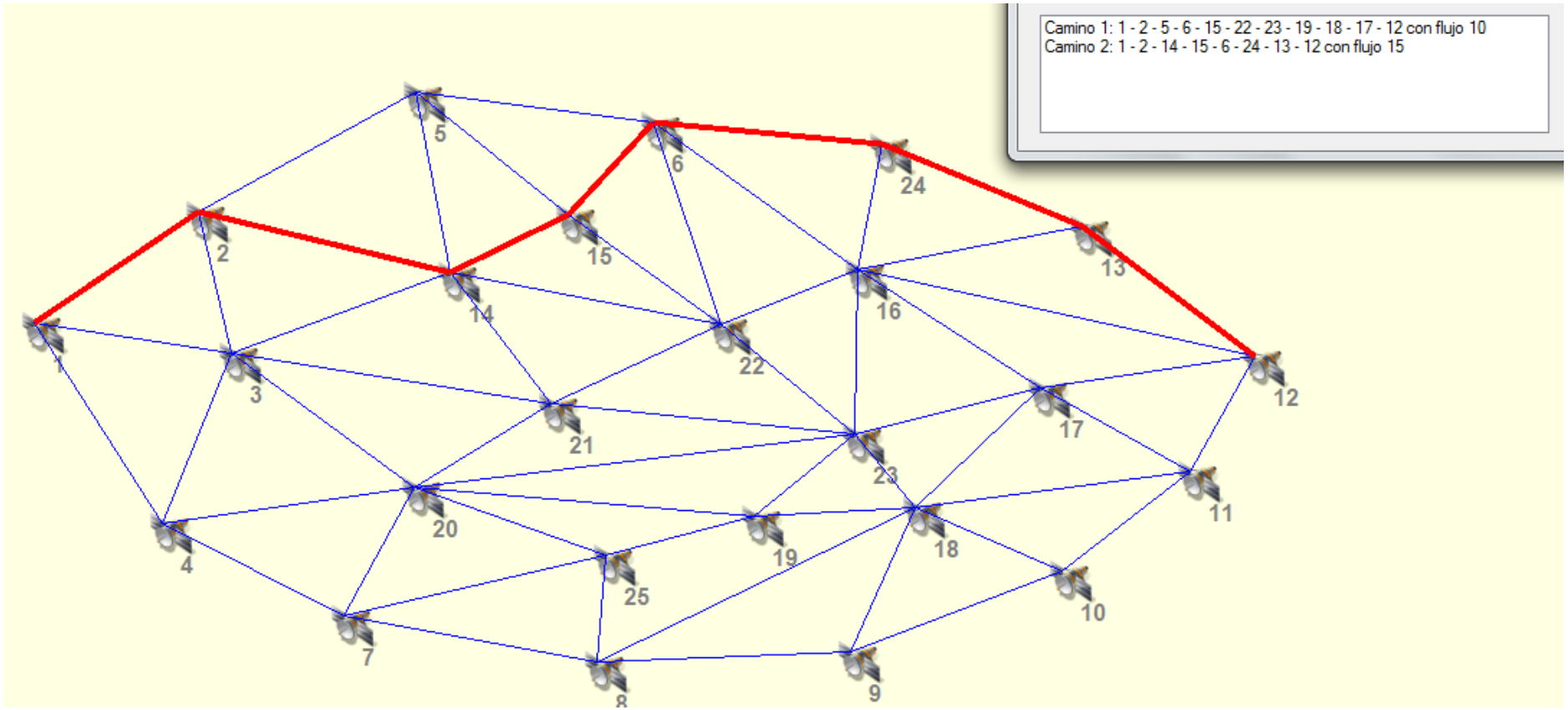

1
2
3
4
5
7
8
9
10
Paso a paso
Siguiente paso
Cerrar
Camino 1: 1 - 3 - 5 - 7 - 8 - 10 con flujo 20
Camino 2: 1 - 2 - 3 - 8 - 7 - 10 con flujo 20
Camino 3: 1 - 3 - 2 - 5 - 8 - 7 - 6 - 9 - 10 con flujo 10
Camino 4: 1 - 4 - 5 - 3 - 6 - 7 - 8 - 9 - 10 con flujo 5
Camino 5: 1 - 4 - 5 - 6 - 8 - 7 - 10 con flujo 5
FIN

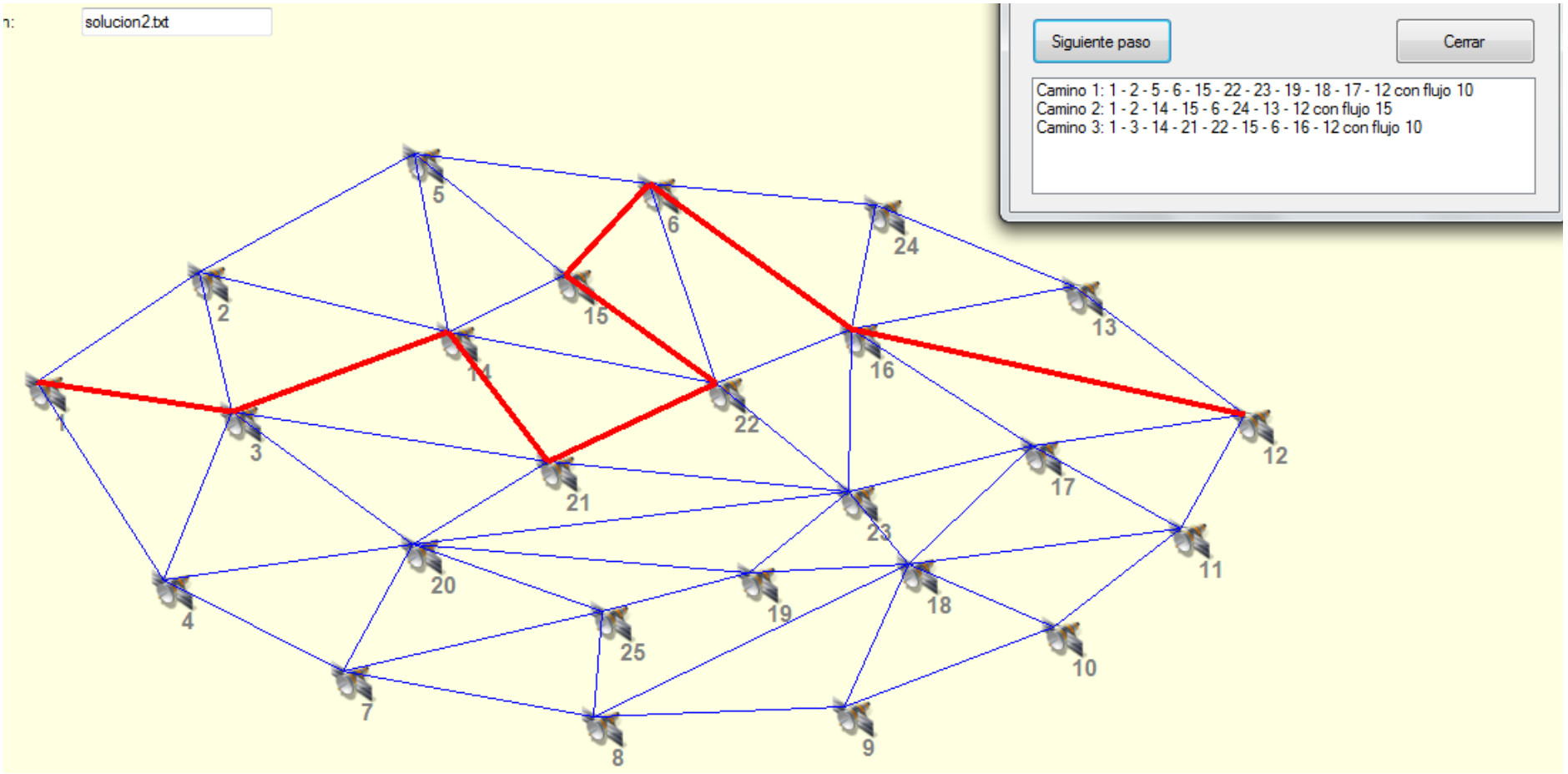
El flujo máximo a través de la red es: 60

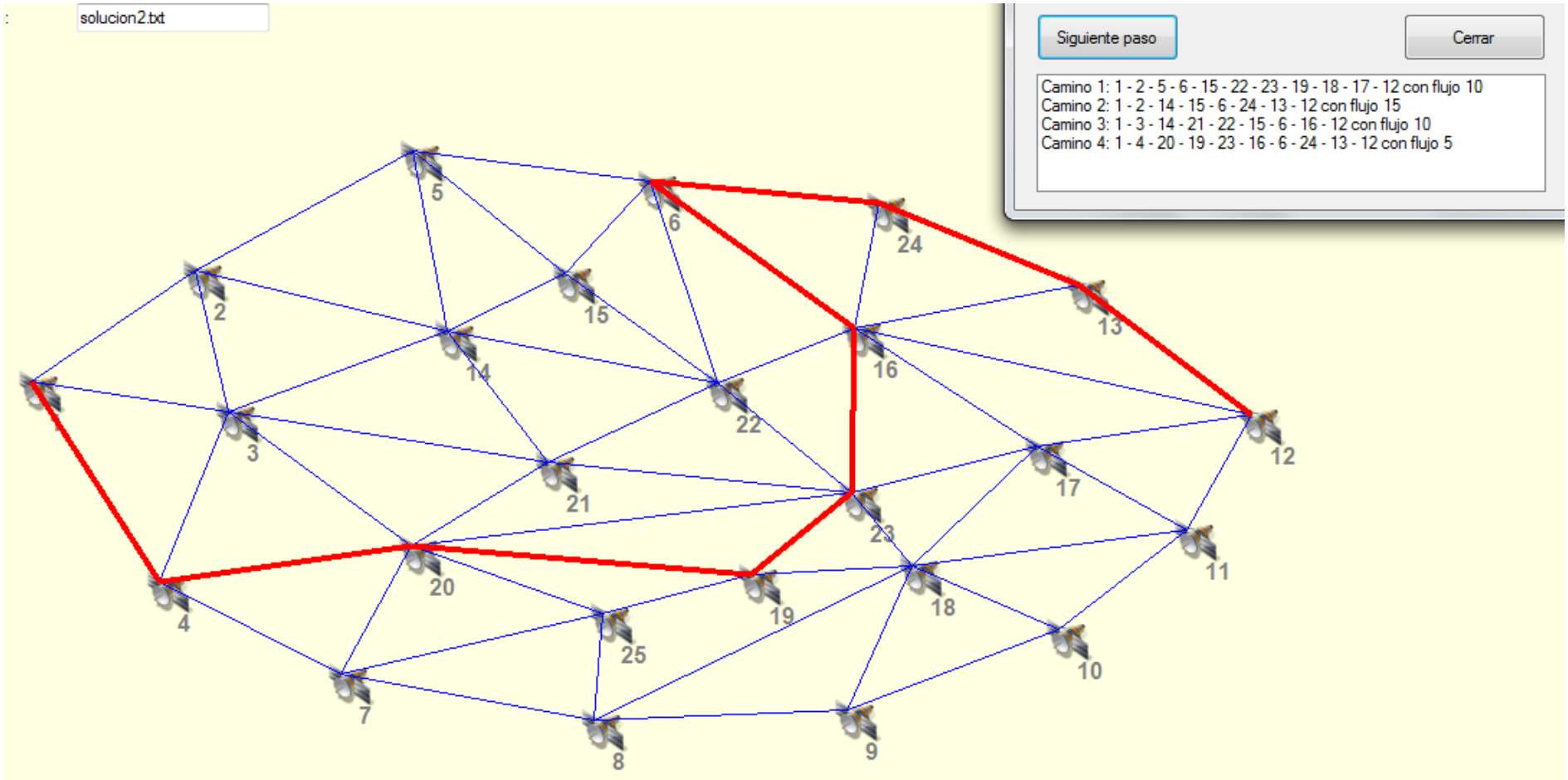
ANEXO 5. RESULTADOS EJEMPLO 2

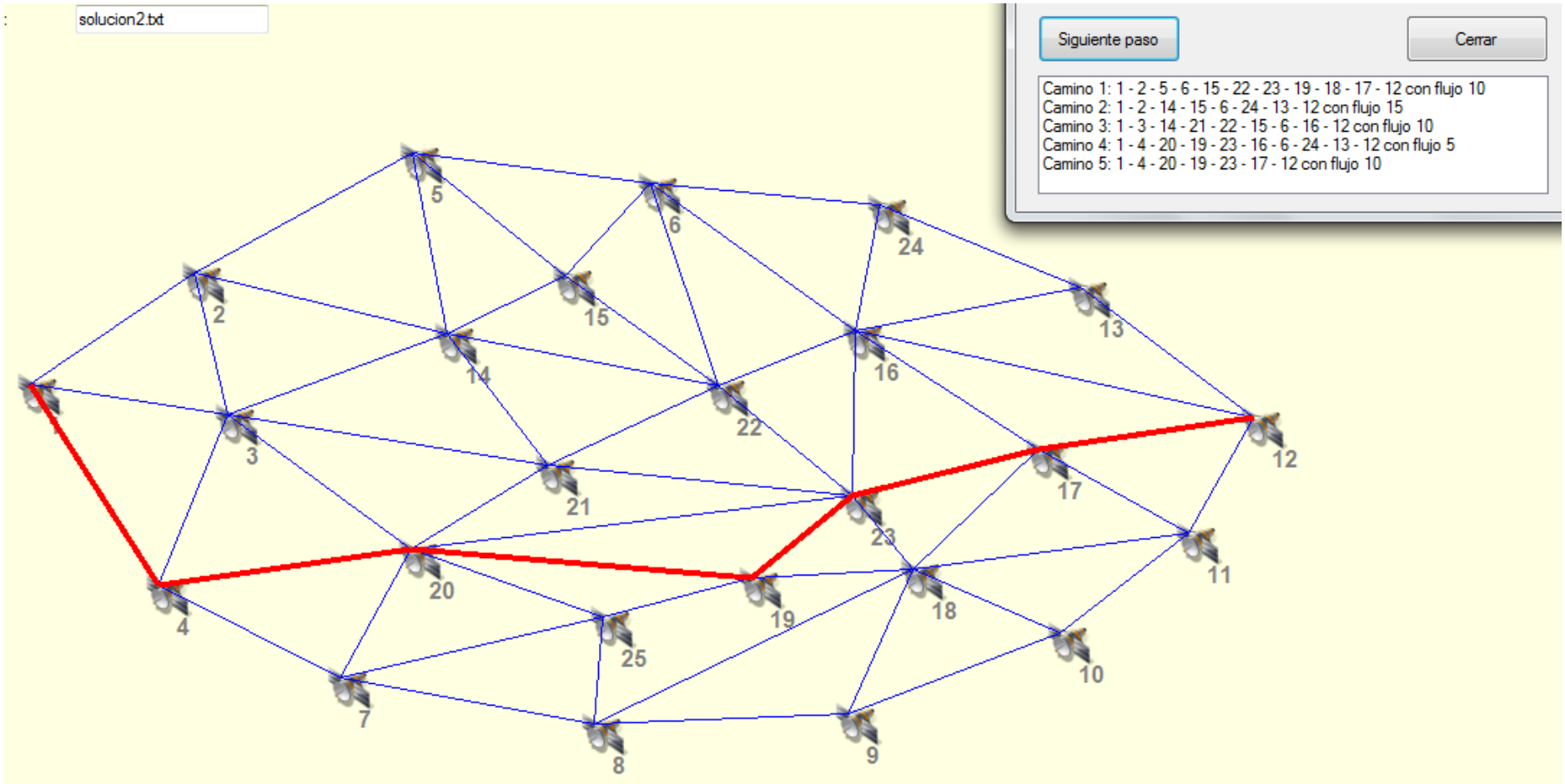


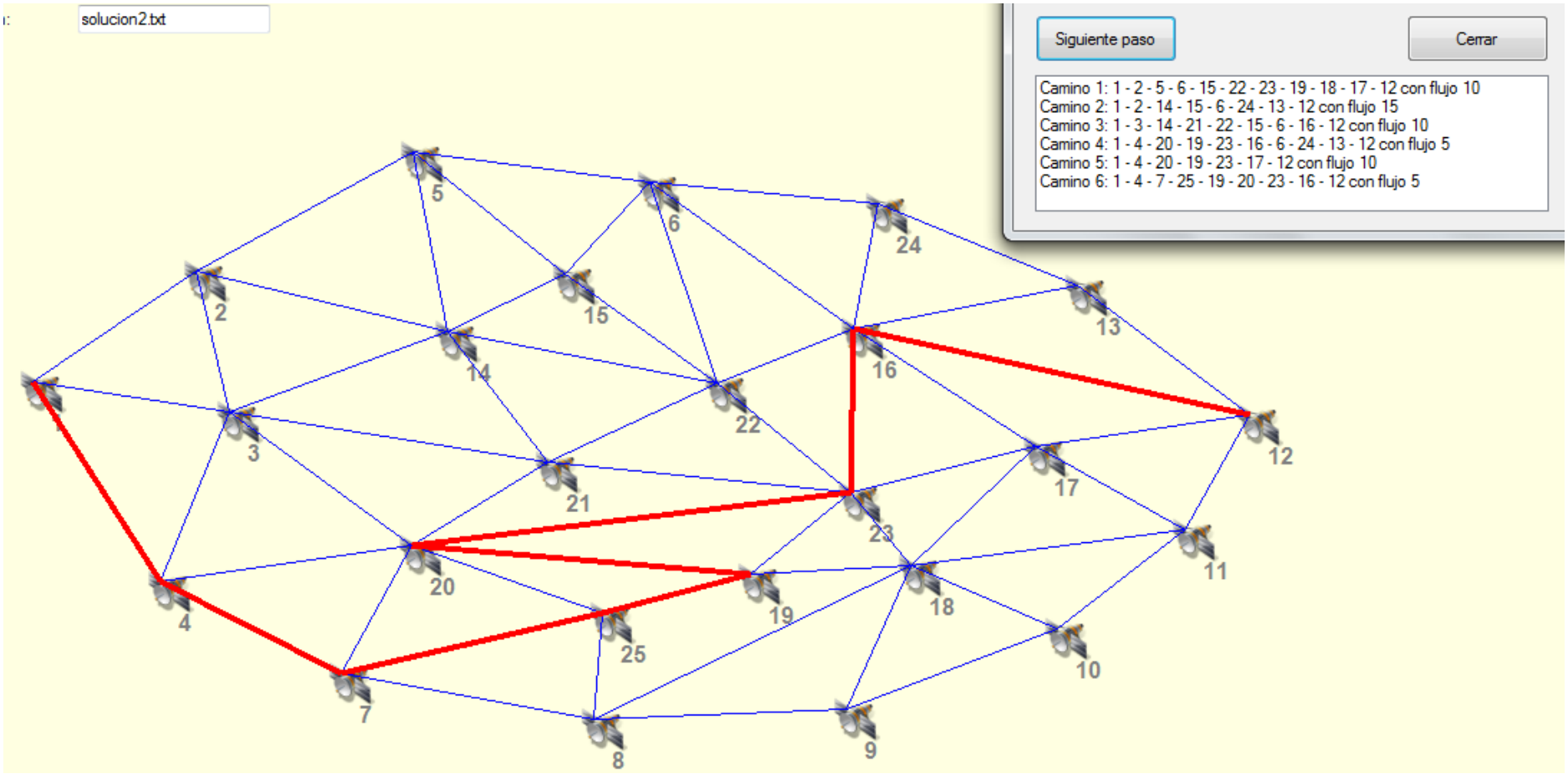

5
6
24
2
15
13
14
16
22
12
3
17
21
23
11
20
18
4
19
25
7
8
Paso a paso
Siguiente paso
Cerrar
Camino 1: 1 - 2 - 5 - 6 - 15 - 22 - 23 - 19 - 18 - 17 - 12 con flujo 10

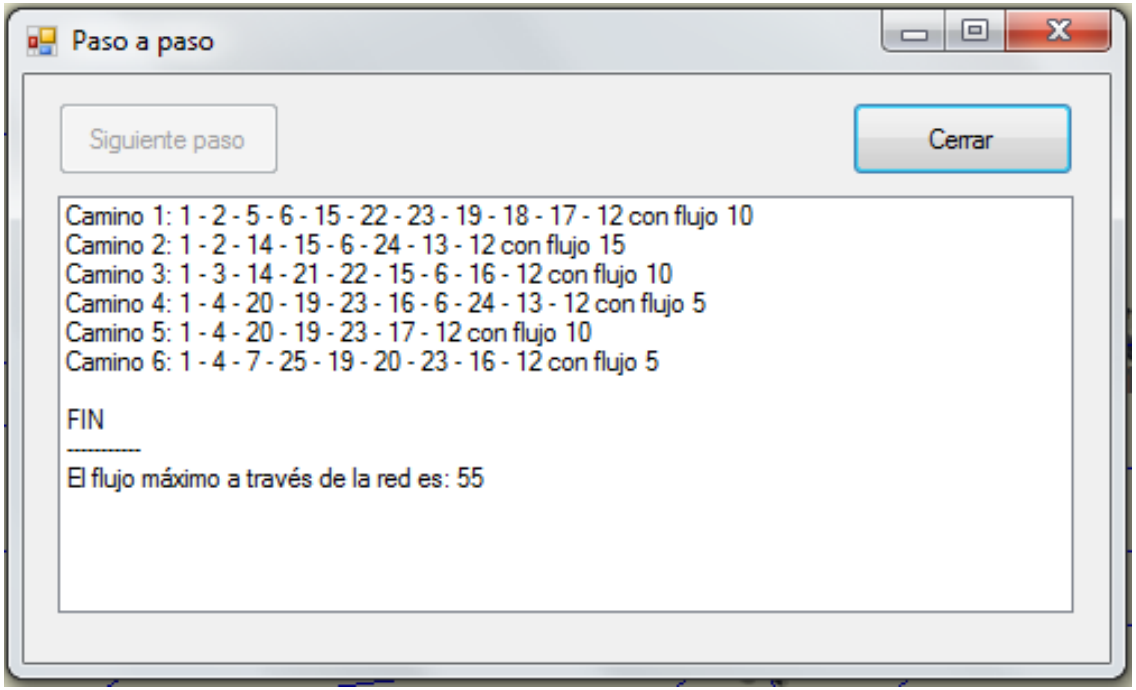

Camino 1: 1 - 2 - 5 - 6 - 15 - 22 - 23 - 19 - 18 - 17 - 12 con flujo 10
Camino 2: 1 - 2 - 14 - 15 - 6 - 24 - 13 - 12 con flujo 15
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25


n:
solucion2.txt
Siguiente paso
Cerrar
Camino 1: 1 - 2 - 5 - 6 - 15 - 22 - 23 - 19 - 18 - 17 - 12 con flujo 10
Camino 2: 1 - 2 - 14 - 15 - 6 - 24 - 13 - 12 con flujo 15
Camino 3: 1 - 3 - 14 - 21 - 22 - 15 - 6 - 16 - 12 con flujo 10
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25


solucion2.txt
Siguiente paso
Cerrar
Camino 1: 1 - 2 - 5 - 6 - 15 - 22 - 23 - 19 - 18 - 17 - 12 con flujo 10
Camino 2: 1 - 2 - 14 - 15 - 6 - 24 - 13 - 12 con flujo 15
Camino 3: 1 - 3 - 14 - 21 - 22 - 15 - 6 - 16 - 12 con flujo 10
Camino 4: 1 - 4 - 20 - 19 - 23 - 16 - 6 - 24 - 13 - 12 con flujo 5
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25


solucion2.txt
Siguiente paso
Cerrar
Camino 1: 1 - 2 - 5 - 6 - 15 - 22 - 23 - 19 - 18 - 17 - 12 con flujo 10
Camino 2: 1 - 2 - 14 - 15 - 6 - 24 - 13 - 12 con flujo 15
Camino 3: 1 - 3 - 14 - 21 - 22 - 15 - 6 - 16 - 12 con flujo 10
Camino 4: 1 - 4 - 20 - 19 - 23 - 16 - 6 - 24 - 13 - 12 con flujo 5
Camino 5: 1 - 4 - 20 - 19 - 23 - 17 - 12 con flujo 10
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25


solucion2.txt
Siguiente paso
Cerrar
Camino 1: 1 - 2 - 5 - 6 - 15 - 22 - 23 - 19 - 18 - 17 - 12 con flujo 10
Camino 2: 1 - 2 - 14 - 15 - 6 - 24 - 13 - 12 con flujo 15
Camino 3: 1 - 3 - 14 - 21 - 22 - 15 - 6 - 16 - 12 con flujo 10
Camino 4: 1 - 4 - 20 - 19 - 23 - 16 - 6 - 24 - 13 - 12 con flujo 5
Camino 5: 1 - 4 - 20 - 19 - 23 - 17 - 12 con flujo 10
Camino 6: 1 - 4 - 7 - 25 - 19 - 20 - 23 - 16 - 12 con flujo 5
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25


Paso a paso
Siguiente paso
Cerrar
Camino 1: 1 - 2 - 5 - 6 - 15 - 22 - 23 - 19 - 18 - 17 - 12 con flujo 10
Camino 2: 1 - 2 - 14 - 15 - 6 - 24 - 13 - 12 con flujo 15
Camino 3: 1 - 3 - 14 - 21 - 22 - 15 - 6 - 16 - 12 con flujo 10
Camino 4: 1 - 4 - 20 - 19 - 23 - 16 - 6 - 24 - 13 - 12 con flujo 5
Camino 5: 1 - 4 - 20 - 19 - 23 - 17 - 12 con flujo 10
Camino 6: 1 - 4 - 7 - 25 - 19 - 20 - 23 - 16 - 12 con flujo 5
FIN

El flujo máximo a través de la red es: 55